The University of Jordan

Authorization Form

I, Heba Mohammad Harahsheh , authorize the University of Jordan to supply copies of my Thesis/ Dissertation to libraries or establishments or individuals on request, according to the University of Jordan regulations.

Signature:

Date: 4/5/2011

الجامعة الأردنيـــة
كلية الدراسات العليا

التاريـــخ: / /

نموذج رقم (١٨)
اقرار والتزام بالمعايير الأخلاقية والأمانة العلمية وقوانين الجامعة الأردنية وأنظمتها وتعليماتها لطلبة الماجستير

أنا الطالب: هبه محمد أحمد الجراجشة الرقم الجامعي: (8080214)

تخصص: علم الحاسوب الكليـــة: الملك عبدالله الثاني لتكنولوجيا المعلومات

عنوان الرسالة: SQL Exception Coverage Using Genetic Algorithms

اعلن بأنني قد التزمت بقوانين الجامعة الأردنية وأنظمتها وتعليماتها وقراراتها السارية المفعول المتعلقة باعداد رسائل الماجستير عندما قمت شخصيا" باعداد رسالتي وذلك بما ينسجم مع الأمانة العلمية وكافة المعايير الأخلاقية المتعارف عليها في كتابة الرسائل العلمية. كما أنني أعلن بأن رسالتي هذه غير منقولة أو مستلة من رسائل أو كتب أو أبحاث أو أي منشورات علمية تم نشرها أو تخزينها في أي وسيلة اعلامية، وتأسيسا" على ما تقدم فانني أتحمل المسؤولية بأنواعها كافة فيما لو تبين غير ذلك بما فيه حق مجلس العمداء في الجامعة الأردنية بالغاء قرار منحي الدرجة العلمية التي حصلت عليها وسحب شهادة التخرج مني بعد صدورها دون أن يكون لي أي حق في التظلم أو الاعتراض أو الطعن بأي صورة كانت في القرار الصادر عن مجلس العمداء بهذا الصدد.

توقيع الطالب:.............................. التاريخ: ٤ / ٥ / ٢٠١١

تعتمد كلية الدراسات العليا
هذه النسخة من الرسالـــة
التوقيع............ التاريخ.............

SQL EXCEPTION COVERAGE USING GENETIC ALGORITHMS

By
Heba Mohammad AL Harahsheh

Supervisor
Dr. Mohammad Al Shraideh

This Thesis was Submitted in Partial Fulfillment of the Requirements for the Master's Degree of Computer Science

Faculty of Graduate Studies
The University of Jordan

May, 2011

COMMITTEE DECISION

This Thesis /Dissertation (**SQL EXCEPTION COVERAGE USING GENETIC ALGORITHMS**) was successfully Defended and Approved on 24/4/2011.

Examination Committee	**Signature**
Dr. Mohammad Alshraideh (Supervisor) Assistant Professor of Software Testing Using Evolutionary Algorithms	------------------------
Dr. Hazem Al Hiary (Member) Assistant Professor of Image Processing and Document Analysis and Recognition	------------------------
Dr. Mohammad Qatawneh (Member) Assistant Professor of Parallel Systems and Networks	------------------------
Dr.Faisal Asad Abu Rub (Member) Assistant Professor of Business Process Improvement building on Process Modeling (Petra University)	------------------------

SQL EXCEPTION COVERAGE USING GENETIC ALGORITHMS

By
Heba Mohammad AL Harahsheh

Supervisor
Dr. Mohammad Al Shraideh

This Thesis was Submitted in Partial Fulfillment of the Requirements for the Master's Degree of Computer Science

Faculty of Graduate Studies
The University of Jordan

May, 2011

COMMITTEE DECISION

DEDICATION

To my parents, brothers and sisters for their patience, endless encouragement and their continuing support.

AKNOWLEDGEMENT

First, I thank Allah (subhanahu wa ta'ala) for helping me to complete this work, and giving me patience and knowledge.

I would like to show my gratitude to my supervisor, Dr. Mohammad Al Shraideh, for his encouragement and guidance during my work.

Heba Harahsheh

Amman 2011

TABLE OF CONTENTS

Contents

LIST OF TABLES

LIST OF FIGURES

TERMINOLOGIES

SQL	Structured Query Language
GAs	Genetic Algorithms
NDF	NO_Data_Found
TMR	Too_Many_Row
HR	Human Resources
TC	Test Cases
GN	Generation Number
GPN	Group Number
DB	Database
PK	Primary Key
PL/SQL	Procedural Language / Structured Query Language

SQL EXCEPTION COVARAGE USING GENITIC ALGORITHM

By
Heba Mohammad Harahsheh

Supervisor
Dr. Mohammad Alshraideh

ABSTRACT

Nowadays Computer Applications used in any filed in our life. Applications differ from each other in complexity and size. So any application needs to check its functionality. Software testing commonly used to detect the presence of errors (failures) in any programs.

In this thesis we generate new technique in software testing, That make test cases automatically to cover three types of exceptions in oracle database: No_Data_Found (NDF), Too_Many_Row (TMR) and Others exceptions. We used genetic algorithm and mutation testing in our proposed idea, we propose fitness function for Genetic Algorithms to do testing of SELECT queries to cover the three types of exceptions, and this fitness function depends on range of data related to each query. The minimum and maximum values for columns in query conditions in the query or its mutant used as a fitness value to guide GA to cover No_Data_Found (NDF), also the aggregation function count of values of columns that related to query conditions used as guided of GA to cover Too_Many_Row (TMR). Our experiments for GA with fitness used human resources (HR) schema in oracle database 10g that installed with oracle.

We also discuss the result of each query and which type of exceptions that covered. And explain the result for each mutant in any queries if it covered the exceptions or have invisible paths. The higher percentage of

coverage exception in our proposed idea (that depends on fitness of queries) for others exceptions then NDF and TMR has lowest percentage.

Finally in this thesis we compare the result with other research that test these types of exceptions without using fitness function in Genetic Algorithms. The results are encouraged and give full coverage for the three exceptions and with high performance than the previous research.

CHAPTER I
Introduction

1.1 Overview

Nowadays many people used computer applications in different areas. While these applications used in business or education or other areas any of these application must be accurate and reliable to save the work of lost for any person used these software. Software testing meeting this goal by makes sure the code does what it designs to do.

This research presents a software technique to generation test cases automatically for testing exceptions handling in database applications. This testing used Genetics Algorithm and mutation testing to cover three types of exceptions in Procedural structural Query Language (PL/SQL) in oracle database, which are No_Data_Found (NDF), Too_Many_Rows (TMR), and Others exceptions.

Software testing is a process used to finding an error for the program or system that executed. This process aimed to evaluate the capability of program and if it result match with its requirement. Some cases in system failed not easy to find the problem like any other physical processes

because it depends on programming languages, operating systems, and hardware platforms the system or programs working on.

Any system almost contains bugs that depend on the complexity of system, and the human skills can’t cover all bugs in the programs with any complexity level. Complete testing infeasible, because the complexity of programs. And discovering the defects in the programs designing also difficult to know because of the same cusses of complexity of system and cannot cover all values to test it so the guarantee correctness of these values is not satisfied.

There are many different ways to find bugs in any program, but these ways are ineffectual with percentage 100%, also testing depends on the complexity of programs. All testing used by tester aimed to get the high level of testing of system. Some time testing integrated in development phases and take about 50% of development time.

Genetic Algorithm (GA) one of the main concepts used for solving different type of problems in many areas of computer science. It’s appeared in United States in the 1970s by John Holland at University of Michigan (Holland,1975). Genetic Algorithm Inspired by the biological evolution

process to find best solution for hard problems specially in searching algorithms, using some operations like, crossover and mutation.

Genetic algorithm used operator called mutation. Mutation means changes that occurred in one or more gene values in chromosome form it initial state. When the changes occurred, new gene value will appear. In SQL mutation new mutants from SQL statement will appear from changes on the original SQL statement, then all mutants generated will add to mutations pool to this SQL statement.

The Genetic Algorithm creates a set of solution (chromosomes or genome) called population. Selection of new population is occurred Depends of fitness (guilty of chromosome) of each chromosome in the population, to have new population better than the old one depends of evaluating of chromosome fitness. In new population (offspring) selection two members (chromosome) depend of their fitness by some methods, Roulette wheel selection is the common one for selection chromosome. Crossover operation is used to exchange (flip) bits of two members that selected in random chosen point to create new mutations. This mechanism is repeated until the target is satisfied like number populations or get of the best solution.

Exceptions occurred when any error or any unusual event in the program appear. Any error can cause exceptions when up normal of program execution flow happened. Exceptions detectable by either hardware or software may require special processing. After detection exceptions in program special that called exceptions handling processes may be required.

Some time exception appears when we try to retrieve data from database and the language of the program that we use deems this case is impossible, and then the exception massage rising. Any programming language contain list of exception with it massage number and exception type for each case of exceptions and exception type as standard in any programming language. Depend of the message of exception the developer can handling the exception by code to cover the exceptions that may be appear in program.

To handling the exceptions there developer covers the exception by built part of code to handling the exception depending of its number. In GA with fitness these is three type of exception will covered using oracle database, there are: No_Data_Found, Too_Many_Row, and Other exceptions.

There are several reasons to get errors in any program. internal exceptions may occur automatically when the program violate any role of database or exceed the limit of the program that runs. Errors caused by hardware failures, network failures, bugs in program, out of memory and other reasons. Any type of exceptions must be handling by giving meaningful message for users to cover all types of exceptions may be raised in run-time. Exception may be definition in two ways: user defined that allow programmer to define his exceptions, Or predefined exceptions that already declared by programming language for most common errors such as divide by zero, out of memory.

1.2 Problem Definition

Exceptions are special case in program that changes the normal flow of program execution. Anything happen under execution and the program can not deal with executing exception handling message will appear to describe the case of the exception. Usually exception caused by design of program or hardware failures, or incorrect input value.

Handling the exception in programming language needed to control the executions of the programs by switch to handling unit in program code when

the exceptions rose. Handling code in any program control any case may be occur in run-time by take suitable actions.

This research depends on the genetic algorithms (GA) for using the fitness of genetic algorithm for testing the exception in oracle database. This research will compared its result with other research Jawabreh (2009), which used genetic algorithms search without guidance. In our work we used mutation testing and genetic algorithm using fitness in genetic algorithm to generate automatic test cases that raising the exceptions.

1.3 Proposed Technique

This research aimed to coverage three types of exceptions: NDF, TMR and Others exceptions automatically. Covering exceptions done by creating test cases using genetic algorithm and mutations for SELECT statement in Oracle database.

Covering three types of exception in our proposed work created using some steps as follows:

1. Insert database query (SELECT statement) and a schema file that describing the structure of the database to system to generate mutations.
2. The system generates a list of SQL mutants for the SELECT query.
3. Using Genetic Algorithm (GA) to generate set of test case to cover any type of the three exceptions in our research.

4. Repeating the process of generating test cases for each mutant of SELECT statement until converging three types of exceptions.

5. The exceptions that remain uncovered after finishing all mutants are considered as invisible paths.

1.4 Organization of the Thesis

This thesis is organized as follows. Chapter II reviews a related work for database testing, Genetic algorithms, exception handling, and mutations testing. In Chapter III we will describe the idea and details of GA with fitness and the implementation of it. Experimental and result discussed in Chapter IV. More details about the result will appear in Chapter V. in Chapter VI discussed Future work and conclusions we have reached.

CHAPTER II

Background and Related Work

2.1 Database Testing Overview

Testing is important areas in research that aimed to reducing number of error that may be appear in any program to produce more reliable systems and have full functional behavior of programs that developed. Testing must cover all queries to have maximum possibility coverage. Covering all cases of quires is difficult depending on data in database and queries itself and used to covering many fault in program. Counting faults in any program is impossible because the fault is ambiguous to defined and diversity of the faults itself. New challenges in software testing appeared by new structural elements in exception handling as propagation paths, exception handling code (Ram Chillarege, 1999).

While software testing used to evaluating capability of program and determining that meets its required results, there are many ways for testing the systems; black-box that derived from functional requirement without know final structure of the program, that some time called data driven or input/output driven. On other hand white-box software under test is visible for tester, and the testing plans are made according to the details of software. This

type of testing some time called class-box testing, logic driven testing or design-based testing.

2.2 Database Application Testing

Many researches in database application testing having high software quality, one of the big trends of modern software engineering is to automate testing activities as much as possible. Automatic testing is cheaper, faster, and less error-prone than humans. And any change in request or configuration of deployment or getting new release of product, still a product attractive in implementation using automatic testing (Carsten, Donald,2008).

Chen, Qiu, Li (2006) using automatic testing by using UML activity diagrams as design specifications, and presented an automatic test case generation approach. Their approach used JAVA program for testing the randomly generates abundant test cases. Jeevarathinam and Antony (2010) used model checkers as test case generation engines. They generate test case by formal model depending on the required software behavior and show the result to determine if the implementation produced the correct output during testing. By using formal specification as the source artifact to generate functional tests for the final product and because the test cases are produced at

an earlier stage in the software development, they are available before the implementation is completed.

Enhance the integration testing of classes by accounting all possible stares of interacting object. Based on intermediate representation named state-activity-diagram, and generate test case to cover state-activity. Zhang et al (2001) generating database instances in test case generation, by considering the semantic of embedded SQL statement; he developed a constrain solver tool that will automatically generates database instances for white box testing of programs.

Chays et al. (2004), generated GENerator for Database Applications (AGENDA) to test database applications, by collecting information's from database schema and application, then the test cases generated automatically.

Testing with source code is a white-box; any mutation testing needs to change and re-write in software code to generate mutations to test flute and bug in original database applications also called white-box. When the change is occur, run the tests if all change is passed then the tests are failed else tests passed.

2.3 Mutation Testing Overview

Mutation testing is time-consuming and complicated if tester test without using tools. The classical approach of mutation testing used a number of versions for each program, each version mutated program to test single fault. Each of these mutation run to test the goal of each version. Every time the test case fail, the mutant called “Killed” and saving the test case until all mutants killed then compare the result with testing the original program. When the tester does more test suites he can be more confident to test his program.

Testing was done by different level for any software to get good quality software, the first level that performed by the programmer called unit testing; this level shows a program passes the test suite but not the quality of the test suite. Mutation testing level measured and improved the quality of test suite.

Jawabreh (2009) generate test cases using genetic algorithm to coverage three type of exceptions handling No_Data_Found (NDF), Too_Many_Rows (TMR) and others exceptions that raised in SELECT statement. He makes experiment under HR schema in oracle database. This GA doesn’t used fitness function it generate random population of the appropriate type within the specified range and all the initial random ten individuals are selected as parents because GA has no fitness function.

W.K. Chan, T.H. Tse (2005) integrated SQL statements and the conceptual data models of an application for fault based testing. Based on the standard types of constraint used in the enhanced entity-relationship model they proposed set of mutation operation. Proposed idea is use the information captured in conceptual data models to facilitate testers to reason about whether a given SQL statement issued by the program unit of a DB application manipulates the correct sets of data. Based on semantic mutation operations on an enhanced entity relationship model and semantic connections between the associated enhanced cross-comparing the database records manipulated by SQL statements and their mutants W.K. Chan, T.H. Tse built there proposed idea for testing of DB application.

Many research worked for testing database using mutation. There are some tools used for this testing like AGENDA. Deng and Chays used AGENDA tools to test relational database application by checking complex properties of the database state transition performed by the transaction under test. Also other tool used for testing database application that SQLMutation (Tuya et al. 2007) that proposed large set of SQL mutation operators for select statement.

Berndt and Watkins (2005) Used genetic algorithm for software testing by combine automated test suite generation techniques with high volume (long sequence) testing. High volume testing repeats test cases many times. These techniques have been found useful for uncovering errors resulting from component coordination problems.

CHAPTER III

Exception Handling Coverage in SQL Database by Generating Test Cases Using Mutation Testing

3.1 System Overview

The aim of this research is covering exceptions by generate test cases using mutation testing. Covering exceptions in PL/SQL database done automatically in our proposed work using mutation testing and genetic algorithm depend on fitness of each query execute in code that built for covering three types of exceptions NDF, TMR and Others exceptions.

3.2 Structure Overview

The system that created in this proposed work implemented using Visual Basic.net 2008 windows Application connected with oracle database 10g. Figure 1 shows the structure of the new system.

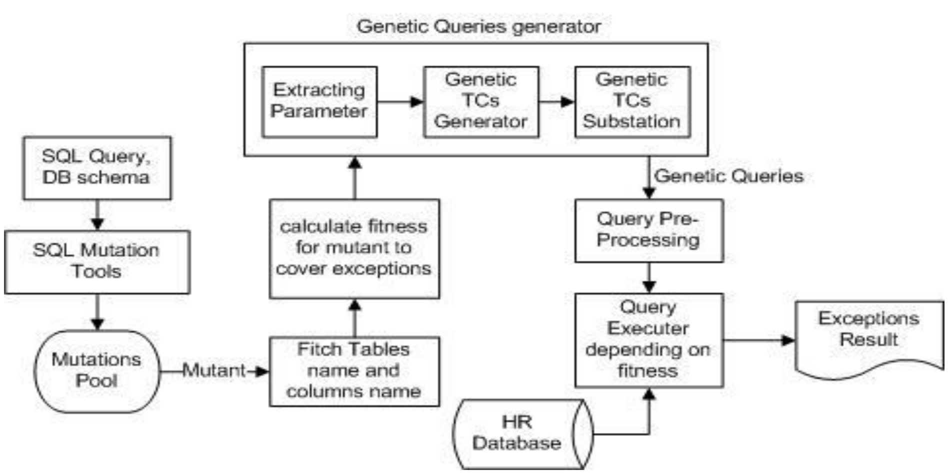

Figure 1 Basic Structure of the New System

HR database schema that appears in Figure 1 which used here is a training database installed with oracle. That we used for our testing. In Windows application system that built in this system used SQL query and DB schema were used as input. This application connects to web application tools called SQL Mutations Tools, this tool generate set of mutants for any query that we use, and save all generated mutants in a pool to use each one in genetic algorithm. Before execute the mutant in GA, the mutant enter to pre-processor steps that fetch tables name and columns name from mutant and make some mortifications, then calculate the fitness of this mutants and the data type for mutant and range of data that related to columns in mutant. In genetic algorithm generate fifty generation each one having ten test cases as individuals. The output of GA sends to pre-processing steps that make some modification in query. The mutant after modifications executed to get the exception covered depending on the fitness and the data in HR database. The outputs of execution are the exceptions that coverage for this mutants.

Tuya et al. (2006) create SQL Mutation Tool. They used set of operations in this tool as standard operation. Following is the description of this tool which is used in this work.

3.3 Genetic Queries Generator

Input value for Genetic queries generator get from previous step. Then it executes queries in three steps and the output send to pre-processing step. Steps for Genetic queries generator are:

3.3.1 Extracting Parameters

In this stage genetic algorithm extract any parameter in mutants enters to this step as input value. Number of parameter in any mutant may be one or more and may be in different datatype, as examples see the next query:

SELECT manager_id from departments where location_id=?xi? or department_id=?yi?

This query has two parameters x and y of type integer, one for location_id and the other for manager_id. The pair of question marks symbol denotes the existence of parameter as it supported in the SQLMutation Tool. The letter just before the last question mark describe the type of parameter where i stands for Integers, d for Decimal, c for Characters and Strings and u for Date data type.

3.3.2 Genetic TC Generator

Generate random population of the appropriate type of parameter in mutant within the specified range of data in database. The initial population consists of ten individuals .The population in the successive generations resulted from crossover and mutation processes. Depending on the fitness function, our genetic algorithm the initial individuals are selected as parents. To get the new offspring a one point cross over operation is performed between each pair of parents. The resulting offspring is then randomly mutated according to their types and they are selected to be the new population in the next generation.

After performing many experiments to find the suitable parameter of generation number, we found that running the genetic algorithm for fifty generations will achieve the desired coverage of the intended exceptions.

3.3.3 Genetic TCs Substitution

The output of previous stage is input of genetic TCs substitution step. In this step the parameter in mutant under process will substitute by the corresponding ones in each of the genetic TCs for fifty generation of queries that belongs to same mutant that is under processing.

3.4 SQL Mutation Tools

Tuya et al. (2006) create SQL Mutation Tool called SQLMutation used for database SELECT queries to generate mutants of SQL SELECT queries. This mutant can create from browser using web interface or from other applications that consuming a web interface. It used SQL query and database schema to generate mutants for query. SQLMutation can access at http://in2test.lsi.uniovi.es/sqlmutation.

SQLMutation used database schema to specify all tables name and columns name for the schema that upload for tools. Declaring schema in SQLMutation done in two ways: Using web interface or using schema in XML which can be built using other tool called XDBSchema.

User generates mutants for any query by upload table schema and write query in the text in web interface/application then set of mutants appear with classifications (Category, Type and subtype). The tool must provide by the data type of column in query, so there is a way for declare data type for system, (?nt?) where n one digit and t indicate data type. There are eight data type: `c` (character string types), `i` (exact numeric integer types), `d` (exact numeric decimal types), `r` (approximate numeric types), `b` (bit types), `t` (time), `u` (date), `v` (timestamp/datetime).

The output of SQLMutation present mutants in table of five columns, below declaration for each column:

1. ID: sequence number.
2. Category: one of SC, OR, NL, IR.
3. Mutant type: the acronym of the mutation operator.
4. Subtype: each of which refer to a particular mutant type when it is applied to a given SQL clause.
5. SQL representing the mutant.

Category in column two of output table contains four types, each one mean type of operation in SQLMutation as following:

1. SC mutations for main SQL clauses
2. OR mutations for operators in the conditions or expressions,
3. NL mutations related for NULL values
4. IR mutations for replacement of identifiers.

Each type of category classified into subtype that identified by three capital letters. Following we will make full description for each type.

- **SC SQL Clause Mutation Operators.**

Contain set of types:

- SEL: SELECT Clause: Where each SELECT or SELECT DISTINCT keywords is replaced by the other.

- JOIN Clause: for each join-type keyword (INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN, and CROSS JOIN) is replaced by each of the others, with making the necessary modifications to ensure the query is syntactically and semantically correct.
- SUB: Subquery Predicates: Subqueries tends to appear in predicates. Different forms of mutation operators designed for this type according to the predicates types ALL,ANY,SOME,IN,NOT IN,EXISTS ,NOT EXISTS.
- GRU: Each of the group by expression is removed, if the removed expression is present in SELECT list then two mutants are generated for each expression, one using the MIN and the other using the MAX aggregate functions. If there is only one group-by-expression, then the whole clause is removed, with making the necessary modifications to ensure the query is syntactically and semantically correct.
- AGR: Aggregate functions: It replaces each aggregate function (MAX, MIN, AVG, AVG(DISTINCT), SUM, SUM(DISTINCT), COUNT, COUNT(DISTINCT)) in a select-list or having-list by each of the others, with making the necessary modifications to ensure the query is syntactically and semantically correct.

- UNI: Query concatenation: Each of the union keywords (UNION, UNION ALL) are replaced by the other keyword and each of the queries that participate in the union are removed.
- ORD: Ordering of the result set. Mutants as the following: the direction of ordering is changed by exchanging between (ASC, DESC) keywords. If neither of these keywords is present, DESC is added. Remove each of the order-by-expressions (if there is only one order-by-expression, then the entire clause is removed) and exchange each pair of adjacent order-by-expressions.

- **OR – Operator Replacement Mutation Operators**

This category operation used to find logical error. It is important operators because in change the output of queries and it appear in condition part of query. It include following types:

- ROR: Relational Operator Replacement: Each operator like $\{=,<>,<,<=,>,>=\}$ is replaced by each of the other operators, by falseop (always returns false 1=0) and by trueop (always returns true 1=1).
- LCR: Logical connector operator: Each of the logical operators (AND, OR) is replaced by each of the other operators, by falseop, by trueop, by leftop (returns the left operand), and by rightop (returns the right operand).

- UOI: Unary Operator Insertion: each arithmetic expression or reference to a number e is replaced by -e, e+1 and e-1.
- ABS: Absolute Value Insertion: Each arithmetic expression or reference to a number e is replaced by ABS(e) and –ABS(e).
- AOR: Arithmetic operator replacement: Each arithmetic operator {+,-,*,/,%} is replaced by each of the others, operators leftop and rightop are applied to the arithmetic expression.
- BTW: Between predicate: Each condition in the form a BETWEEN x AND y is replaced by a>x AND a<=y and by a>=x AND a<y. If the condition is NOT BETWEEN, the mutants are negated.
- LKE: Like predicate: The mutations will be restricted to exercising the behavior of the wildcards {%,_} (the percent symbol means for any character string and the underscore means for an individual character). Each wildcard is mutated by removing the wildcard, replacing the wildcard by the other, removing the character just before the wildcard if it is not at the beginning of s and removing the character just after the wildcard if it is not at the end of s. adding each of the wildcards at the beginning If no wildcard is present at the beginning of s, and adding each of the wildcards at the end and if no wildcard is present at the end of s.

- **NL – NULL Mutation Operators**

Used for NULL values that may retrieve from database. Following types relater to NL:

  - NLF: Null check predicates: Each occurrence of one of the predicates IS NULL or IS NOT NULL is replaced by the other.
  - NLS: Null in select list: This operator aim to produce NULL values in the output variables; it replaces references to column c by if null(c,r) function which substitutes the value c by r whenever c is null.
  - NLI/NLO: Nulls in the input data. Conditions in SQL can have three possible values true, false, and null ,the NLI and NLO operators handle the situations when output of a condition is NULL:
    - NLI: Include nulls: This operator forces a true value of the condition when there is a null value. For each attribute a in a condition C of the form a R b or b R a, where R is relational operator {=,<>,<,<=,>,>=}, the condition is replaced by C OR a IS NULL.
    - NLO: Other nulls: For each attribute a in C, the condition is replaced by, NOT C OR a IS NULL, a IS NULL, and a IS NOT NULL.
    -

- **IR – Identifier Replacement Mutation Operators**

  Operation used to discover error in queries:

  - IRC: Column replacement: Each column reference is replaced by each of the other column references, constants and parameters that are present in the query and are type compatible.
  - IRT: Constant replacement: Each constant is replaced by each of the other constants, columns and parameters that are present in the query and are type compatible.
  - IRP: Parameter replacement: Each query parameter reference is replaced by each of the other parameters, columns and constants that are present in the query and are type compatible.
  - IRD: Hidden column replacement: Each column attribute reference is replaced by each of the other columns that are defined in its table provided that they have not been the replacement in any of the other IR operators and are type compatible.

CHAPTER IV

Experiments and Results

In this chapter we will describe database schema and SQL queries that we used in this approach and testing using it.

4.1 Database Schema

The experiment executed under Human Recourses (HR) database schema. This database schema contains seven tables: countries, departments, employees, jobs, job_history, locations and regions. HR schema installed automatically with oracle 10g.

Figure 2 show HR database diagram. It shows also the tables in the HR schema and the columns in each table, as well as dependencies between these tables. The employees table has columns employee_id (primary key), first_name, last_name, email, phone_number, hire_date, job_id, salary, commission_pct, manager_id, and department_id. Each employee must be related to one department in the departments table and one job in the jobs table and may be related to one or more records in the job_history table.

The jobs table has the columns job_id (primary key), job_title, min_salary, and max_salary. Each job may be related to one or more employees in the employees table. The job_history consists of the following columns: employee_id (primary key), start_date (primary key), end_date, job_id, and

department_id. Each job history record must be associated with one employee in the employees table.

The countries table consists of the following columns: country_id (primary key), country_name and region_id. Each country may be related to one or more locations and must be associated with one region.

The regions table has columns region_id (primary key) and region_name. Each region may be associated with one or more countries the locations table has columns location_id (primary key), street_address, postal_code, city, state_province, and country_id. Each location may be associated with one or more departments and must be associated with one country.

The departments table has columns department_id (the primary key), department_name, manager_id, and location_id. Each department must be related to one or more employees in the employee table and must be related to one location in the locations table (Oracle).

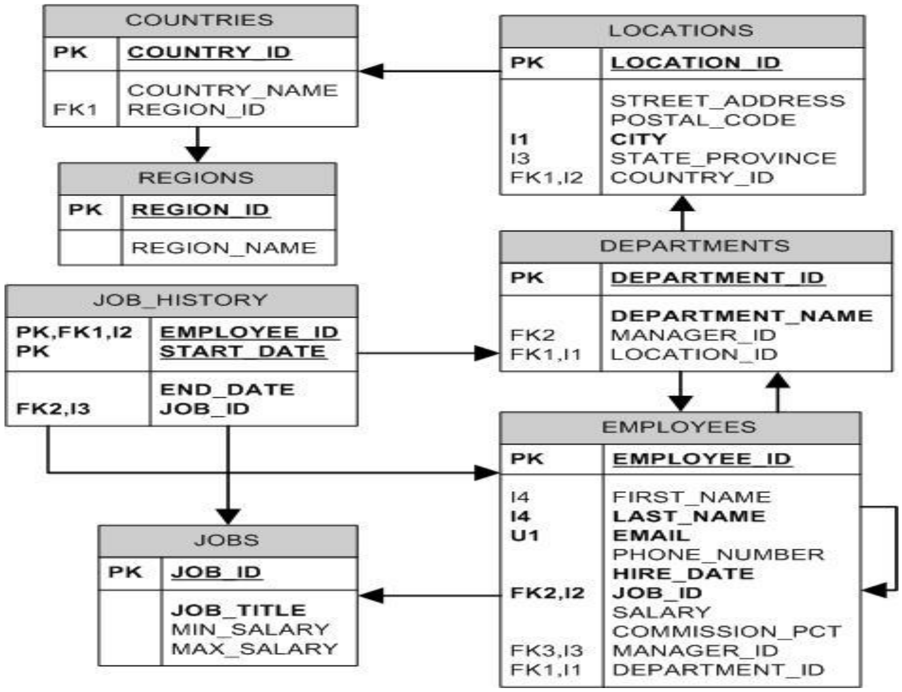

Figure 2 HR Database Diagram

4.2 HR Schema

```
<schema dbms="Oracle" schema="HR">
        <table name="COUNTRIES">
        <column name="COUNTRY_ID" type="CHAR" size="2" key="true"
        notnull="true" />
        <column name="COUNTRY_NAME" type="VARCHAR2" size="40"
        />
        <column name="REGION_ID" type="NUMBER" size="22"
        fk="REGIONS.REGION_ID" fkname="COUNTR_REG_FK" />
</table>
        <table name="DEPARTMENTS">
        <column name="DEPARTMENT_ID" type="NUMBER" size="4"
        key="true" notnull="true" />
        <column name="DEPARTMENT_NAME" type="VARCHAR2"
        size="30" notnull="true" />
        <column name="MANAGER_ID" type="NUMBER" size="6"
        fk="EMPLOYEES.EMPLOYEE_ID" fkname="DEPT_MGR_FK" />
        <column name="LOCATION_ID" type="NUMBER" size="4"
        fk="LOCATIONS.LOCATION_ID" fkname="DEPT_LOC_FK" />
```

```
</table>
<table name="EMPLOYEES">
    <column name="EMPLOYEE_ID" type="NUMBER" size="6"
    key="true" notnull="true" />
    <column name="FIRST_NAME" type="VARCHAR2" size="20" />
    <column name="LAST_NAME" type="VARCHAR2" size="25"
    notnull="true" />
    <column name="EMAIL" type="VARCHAR2" size="25"
    notnull="true" />
    <column name="PHONE_NUMBER" type="VARCHAR2" size="20"
    />
    <column name="HIRE_DATE" type="DATE" notnull="true" />
    <column name="JOB_ID" type="VARCHAR2" size="10"
    notnull="true" fk="JOBS.JOB_ID" fkname="EMP_JOB_FK" />
    <column name="SALARY" type="NUMBER" size="8,2" />
    <column name="COMMISSION_PCT" type="NUMBER" size="2,2"
    />
    <column name="MANAGER_ID" type="NUMBER" size="6"
    fk="EMPLOYEES.EMPLOYEE_ID"
    fkname="EMP_MANAGER_FK" />
    <column name="DEPARTMENT_ID" type="NUMBER" size="4"
    fk="DEPARTMENTS.DEPARTMENT_ID"
    fkname="EMP_DEPT_FK" />
</table>
<table name="JOB_HISTORY">
    <column name="EMPLOYEE_ID" type="NUMBER" size="6"
    key="true" notnull="true" fk="EMPLOYEES.EMPLOYEE_ID"
    fkname="JHIST_EMP_FK" />
    <column name="START_DATE" type="DATE" key="true"
    notnull="true" />
    <column name="END_DATE" type="DATE" notnull="true" />
    <column name="JOB_ID" type="VARCHAR2" size="10"
    notnull="true" fk="JOBS.JOB_ID" fkname="JHIST_JOB_FK" />
    <column name="DEPARTMENT_ID" type="NUMBER" size="4"
    fk="DEPARTMENTS.DEPARTMENT_ID"
    fkname="JHIST_DEPT_FK" />
</table>
<table name="JOBS">
    <column name="JOB_ID" type="VARCHAR2" size="10" key="true"
    notnull="true" />
```

```
        <column name="JOB_TITLE" type="VARCHAR2" size="35"
        notnull="true" />
        <column name="MIN_SALARY" type="NUMBER" size="6" />
        <column name="MAX_SALARY" type="NUMBER" size="6" />
        </table>
        <table name="LOCATIONS">
        <column name="LOCATION_ID" type="NUMBER" size="4"
        key="true" notnull="true" />
        <column name="STREET_ADDRESS" type="VARCHAR2"
        size="40" />
        <column name="POSTAL_CODE" type="VARCHAR2" size="12" />
        <column name="CITY" type="VARCHAR2" size="30" notnull="true"
        />
        <column name="STATE_PROVINCE" type="VARCHAR2" size="25"
        />
        <column name="COUNTRY_ID" type="CHAR" size="2"
        fk="COUNTRIES.COUNTRY_ID" fkname="LOC_C_ID_FK" />
</table>
<table name="REGIONS">
        <column name="REGION_ID" type="NUMBER" size="22"
        key="true" notnull="true" />
        <column name="REGION_NAME" type="VARCHAR2" size="25" />
</table>
        <table name="EMP_DETAILS_VIEW" type="view">
        <column name="EMPLOYEE_ID" type="NUMBER" size="6"
        notnull="true" />
        <column name="JOB_ID" type="VARCHAR2" size="10"
        notnull="true" />
        <column name="MANAGER_ID" type="NUMBER" size="6" />
        <column name="DEPARTMENT_ID" type="NUMBER" size="4" />
        <column name="LOCATION_ID" type="NUMBER" size="4" />
        <column name="COUNTRY_ID" type="CHAR" size="2" />
        <column name="FIRST_NAME" type="VARCHAR2" size="20" />
        <column name="LAST_NAME" type="VARCHAR2" size="25"
        notnull="true" />
        <column name="SALARY" type="NUMBER" size="8,2" />
        <column name="COMMISSION_PCT" type="NUMBER" size="2,2"
        />
        <column name="DEPARTMENT_NAME" type="VARCHAR2"
        size="30" notnull="true" />
```

```
        <column name="JOB_TITLE" type="VARCHAR2" size="35"
        notnull="true" />
        <column name="CITY" type="VARCHAR2" size="30" notnull="true"
        />
        <column name="STATE_PROVINCE" type="VARCHAR2" size="25"
        />
        <column name="COUNTRY_NAME" type="VARCHAR2" size="40"
        />
        <column name="REGION_NAME" type="VARCHAR2" size="25" />
</table>
</schema>
```

4.3 SQL Queries

Seventeen different SELECT statements related to HR database that we used in this research. These queries include conditions like: WHERE, HAVING, and ON. The queries that contain WHERE condition may implemented as: [not] BETWEEN, [not] IN, [not] LIKE, IS [not] NULL, logical connector AND and OR and the using of expressions with relational operators like (=, >, <, <=, >=, <>).

Our sample queries consider different ways according to a SELECT statement as Follows:

- The select clause may contain [all] column[s], expressions, string, alias and all functions types.
- The form clause may use one or more tables, outer join may be used, if outer join used query converted manually to WHERE clause.
- The keywords like: group by, and order by may be used.

- Different data type parameters and constants in the queries are used.

4.4 Query List

Following are list of queries that are used in our experiments :

1. SELECT * from employees where employee_id=?xi?
2. SELECT (employee_id) from employees where hire_date>?xu? order by department_id
3. SELECT manager_id from departments where location_id=?xi? or department_id=?di?
4. SELECT e.employee_id,j.max_salary from employees e, jobs j where e.job_id=j.job_id and e.commission_pct<=?xd?
5. SELECT e.first_name from employees e left join job_history h on e.employee_id=
   h.employee_id and h.start_date=?xu?
6. SELECT job_id,avg(salary) from employees group by job_id having sum(salary)<?xi?
7. SELECT job_id from job_history where end_date between ?xu? and ?yu?
8. SELECT region_name from countries c , regions r where r.region_id=c.region_id and c.country_id=?xc?
9. SELECT job_title from jobs where (max_salary + min_salary)/2 <?xi?

10. SELECT postal_code from locations where location_id in(?ai?,?bi?,?ci?)
11. SELECT first_name||last_name from employees where commission_pct * 100>?xi?
12. SELECT location_id from locations where state_province is not null and street_address=?xc?
13. SELECT * from departments d,locations l where d.location_id=l.location_id and l.city like ?xc?
14. SELECT job_title,max(salary) from employees e, jobs j where e.job_id=j.job_id group by job_title having max(salary)>?xi?
15. SELECT count(*) from employees where department_id=?xi?
16. SELECT * from departments where manager_id is null and location_id=?xc?
17. SELECT region_id, count(country_id) from countries where region_id>?xi? group by region_id having count(country_id)<?yi? order by region_id desc

4.5 Experiments

All of seventeen queries that choose for testing GA with fitness are execute. First generate number of mutants for each SELECT query using tool

named SQLMutation accessible at http://in2test.lsi.uniovi.es/sqlmutation (Tuya et al., 2006). Then send mutations of queries one by one to genetic queries generator. All the results are then executed to check the area of exceptions coverage. Three classifications group appear depend on the area exception coverage:

1. Group 1: consists of mutants whose genetic queries cover all types of exceptions in the predefined generation's number.
2. Group 2: The invisible path exceptions group, which consists of mutants whose genetic queries, is impossible to cover some types of exceptions in the predefined generation number due to the nature of the mutant or a database constraint.
3. Group 3: consists of mutant whose genetic queries could raise some type of exceptions, but genetic queries generator stopped before this happens, because the maximum generation number parameter was reached.

Each one of the following table contain the result of the new system GA with fitness for seventeen queries. Each table contains details of executions result that are:

- Mutant ID: The identification number of each mutant .We uses the same IDs that are generated from SQL mutation Tool (Tuya et al., 2006).
- Mutant Subtype: Refers to type of mutants when it is applied to particular SQL clause (as discussed in chapter III).
- Exceptions: Contains the three types of exceptions. No_Data_Found (NDF), Too_Many_Rows (TMR) and **Others**. For each exception type we show if it is covered or not with the generation number (GN) .The generation number parameter was set to be fifty.
- GPN: The number of the group to which each mutant belongs according to its exception coverage area as it previously explained.

For More details about the classification of mutants and the mutated SQL statements see Appendix A.

In GA with fitness that we built in this work used to cover three type of exception as appeared before. Our experiments used this approach for seventeen queries, and the result for all queries shown in Appendix C. this research used fitness function in GA for testing coverage three types of exception. Maximum and minimum value used as fitness to test coverage of NDF exception, if the value that generated by GA for column in the query or mutant under testing greater than the maximum value of this column in the

database or less than the minimum value then NDF exception converge to this query. Fitness for TMR is COUNT for the value that generated to column for query, if the value that generated appears two times or greater and that depending of structure of query then TMR exception coverage. In our approach covering exception contain only the three types. Below is an outline of exceptions handling structure with example of Query 1:

```
DECLARE
Declaration section
BEGIN
SELECT * from employees where employee_id=?xi?
…………
 EXCEPTION
WHEN NO_DATA_FOUND THEN
        Error handling statements
WHEN TOO_MANY_ROWS THEN
        Error handling statements
WHEN OTHERS THEN
        Error handling statements
END;
```

Table 1 show the result of our experiment for query 1 using GA with fitness. In query 1 employee_id column contain range of data in HR schema. Depending in our approach now we will use query 1 as example of this

approach. Minimum value for employee_id in HR schema is 100 and maximum 206. So if GA generates value for Employee_id 300, NDF exception will coverage because the value that generated larger than maximum value for employee_id column and the same if value generated less than minimum value. But in this query TMR exception con not coverage TMR exception because employee_id is a primary key and there is no duplicated for any value so the count of each value in the range of employee_id value in schema in any time equal 1. But when the SQLMutation tool generates mutants for this query and makes some changes in its structure, TMR exception coverage as Table 1 below shown.

After we generate a pool of mutants for query 1, each mutant entered to GA with Fitness and execute after generate many values for many test cases. For example mutant 6 covered NDF exception from first TCs also others exception covered from first generation, but TMR exception not covered and impossible to covered because the converge exception depend on query structure and data in database schema in mutant 6:

SELECT * FROM employees WHERE (-(employee_id)) = ?xi?

In HR schema employee_id always positive so TMR con not converge any time. Mutant 6 related to group 2 that mean impossible to cover because mutant has invisible paths. On other hand some mutant that related to query 1

covered all types of exception early as mutant 13 that need 8 TCs to covered NDF exception, but NDF and Others exceptions covered early form first TCs. So Mutant 13 related to group 1 as table appear because mutant 13 covered all types of exceptions in less than 50 TCs.

Query 1

SELECT * from employees where employee_id=?xi?

Table 1 Query 1 Mutant's Exceptions Coverage

Mutant ID	Mutant Subtype	Exceptions						GPN
		NDF		TMR		Others		
		GN	Coverage	GN	Coverage	GN	Coverage	
2	ABSW	1	y	50	n	1	y	2
3	ABSW	1	y	50	n	1	y	2
4	UOIW	1	y	50	n	1	y	2
5	UOIW	1	y	50	n	1	y	2
6	UOIW	1	y	50	n	1	y	2
7	IRDDW	1	y	10	y	1	y	1
8	IRDDW	1	y	50	n	1	y	2
9	IRDDW	1	y	8	y	1	y	1
10	IRDDW	1	y	20	y	1	y	1
11	RORW	50	n	1	y	1	y	2
12	RORW	1	y	15	y	1	y	1
13	RORW	8	y	1	y	1	y	1
14	RORW	1	y	14	y	1	y	1
15	RORW	16	y	1	y	1	y	1

CHAPTER V

Results Discussion

In this chapter we will discussion the result of the proposed system, by showing the result of coverage the three types of exceptions for all seventeen queries that we used. Then we will show the analyses of result for each query that we used in this research. Finally we will compare our result with other work that used mutation testing to generate test cases for exception handling without guidance for GA

Query 1

select * from employees where employee_id=?xi?

Coverage for Three Types of Exceptions

As shown in Figure 3 the coverage the three types of exceptions for query 1 that NDF, TMR and Others exceptions. Others exceptions converge has the highest percentage with values 100%. NDF exception has 93% but TMR has 57% with lowest ratio this is because TMR exception has large number of invisible path than others.

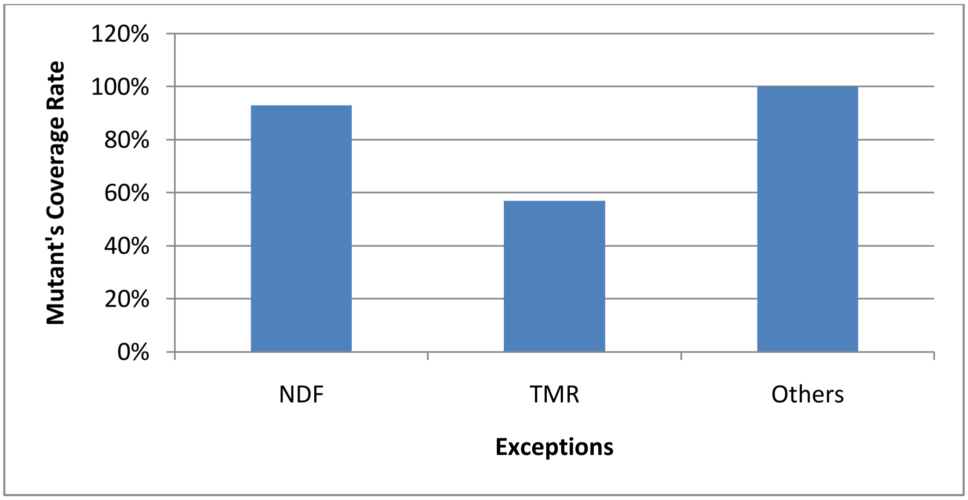

Figure 3 Total Mutants Exceptions Coverage in Query 1

Group One Mutation Coverage

Figure 4 shows Group one and all types of exceptions converge on it. x axis shows the mutants number of query 1 and its subtype that appear in Group 1. M12, M13, M14 and M15 of type Operator Replacement (OR). In M12 and M14 that used operations (>,>=) depend on our sample data that we use it need small number to cover TMR exception so it appear later because genetic algorithm generate small number in later stage. In same time using operations (<, <=) in mutants M13 and M15 with generate small value late that make NDF also appear late.

Identified Replacement (IR) type appears in M7, M9 and M10. TMR covered later for three mutants because the generation value for three mutants has low probability to generate suitable values for each column depend on it

for conditions of each mutants i.e. in M7 need to generate large number for salary column that is between (2100-2400), but in M9 and M10 need to generate small number that the maximum number is 205 for manager_id, and the maximum number for department_id is 110. And the probability of generate this number in genetic TCs is low.

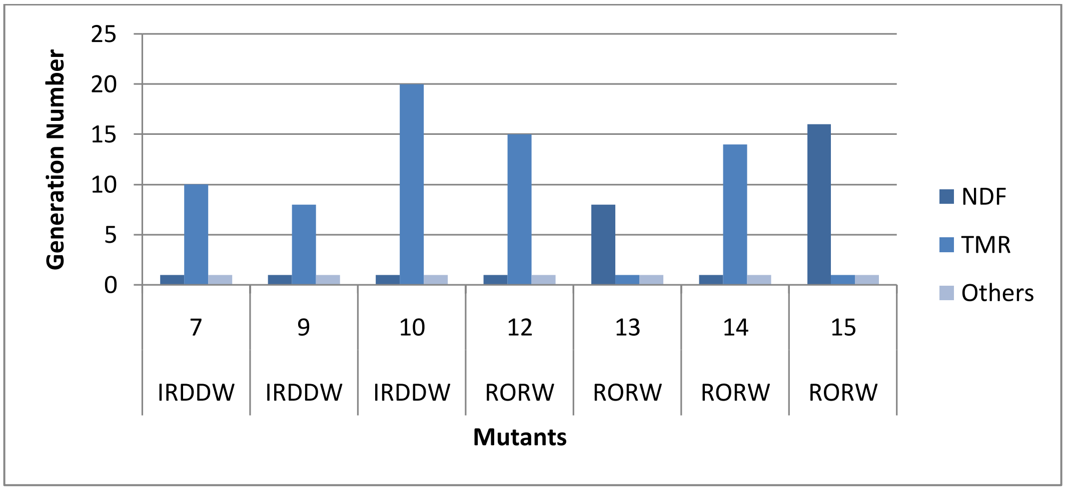

Figure 4 Group 1 Mutants Coverage in Query 1

Group Two Mutants Coverage

In each mutant that genetic query is trying to cover exceptions but we found that is impossible to cover some type of exceptions depending on database constraint and mutant type. Table 18 shows the mutants that related to group two while it has a lot of invisible paths.

Table 2 Group 2 Mutants Coverage in Query 1

Mutants	Invisible Paths	Reasons
M2,M4,M5	TMR	Mutants have condition about employee_id which is primary key (PK).
M3,M6	TMR	Employee IDs in schema are positive.
M8	TMR	Range of data for commission_pct is very small and there is no variety of data and generate later in genetic algorithm.
M11	NDF	The employee_id is a sequence number in employee table and PK, and having large number of employee.

Query 2

select employee_id from employees where hire_date>?xu? order by department_id

Coverage for Three Types of Exceptions

Figure 5 show that all types of exceptions converge in query 2 with high percentage for others exceptions. Also NDF and TMR close to others

exceptions percentage, that mean the invisibility paths are not appeared so much in this mutants of query 2.

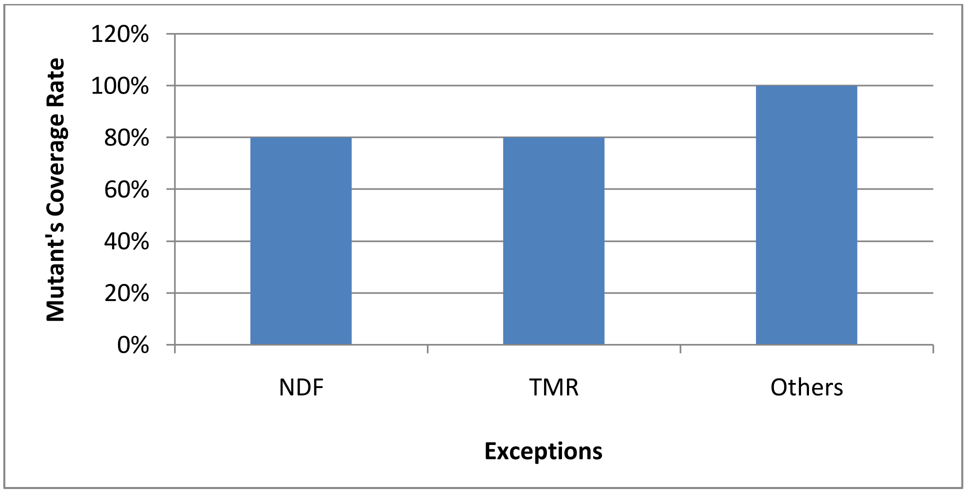

Figure 5 Total Mutants Exceptions Coverage in Query 2

Group One Mutation Coverage

Figure 6 shows that only one subtype appears in group one. each one in M13, M14 and M15 appear from first generation. RORW subtype related to OR operation that can replace operation in original query < with operations in M13, M14 and M15 that (<, >=, <=). All exceptions covered early in these mutants from first generations.

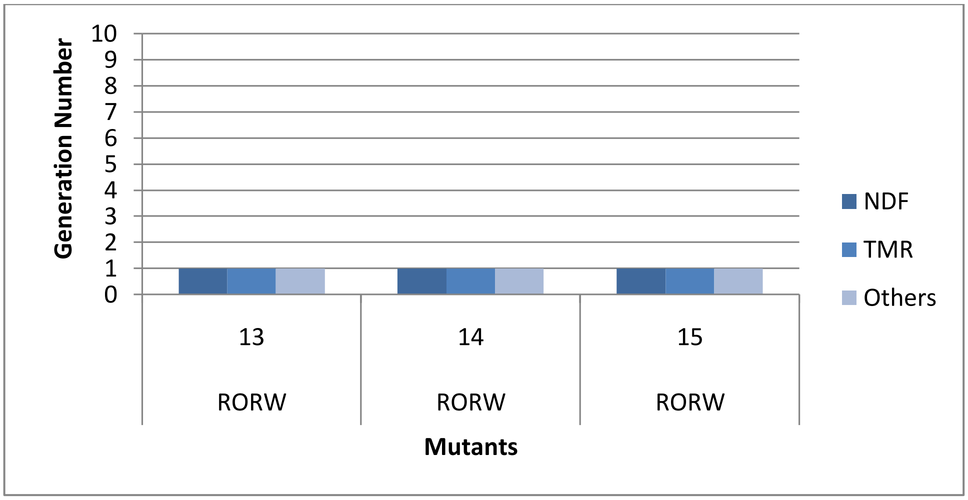

Figure 6 Group 1 Mutants Coverage in Query 2

Group Two Mutation Coverage

In M12 replace > operation from original query to <>, that make invisible path because the hire_date in employees table contain many values and <> operation make the path for data invisible path.

Table 3 Group 2 Mutants Coverage in Query 2

Mutants	Invisible Paths	Reasons
M12	TMR	There are many values in employees table for hire_date.

Query 3

SELECT manager_id from departments where location_id=?xi? or department_id=?xi?

Coverage for Three Types of Exceptions

In Figure 7 shows that the lowest percentage appears for TMR exceptions because there is only some cases cover TMR exceptions for this query. But in table 20 there is many values related to group 3 that means if generate over of fifteen generation for this query the coverage of this exception will be occurred.

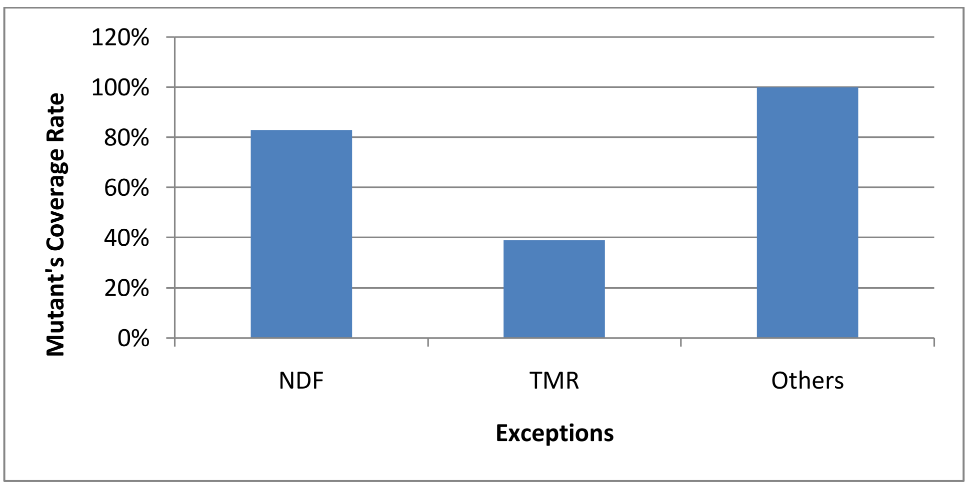

Figure 7 Total Mutants Exceptions Coverage in Query 3

Group One Mutation Coverage

Figure 8 shows that how all type of exceptions covered early for M24,M25,M26 and M27, also Others and NDF covered early in M16 but

TMR appears later because the operation = replace with (<,<=,>,>=) in previous mutants that covered early. Depending on the fitness of query and the range of values that related to column selected from database.

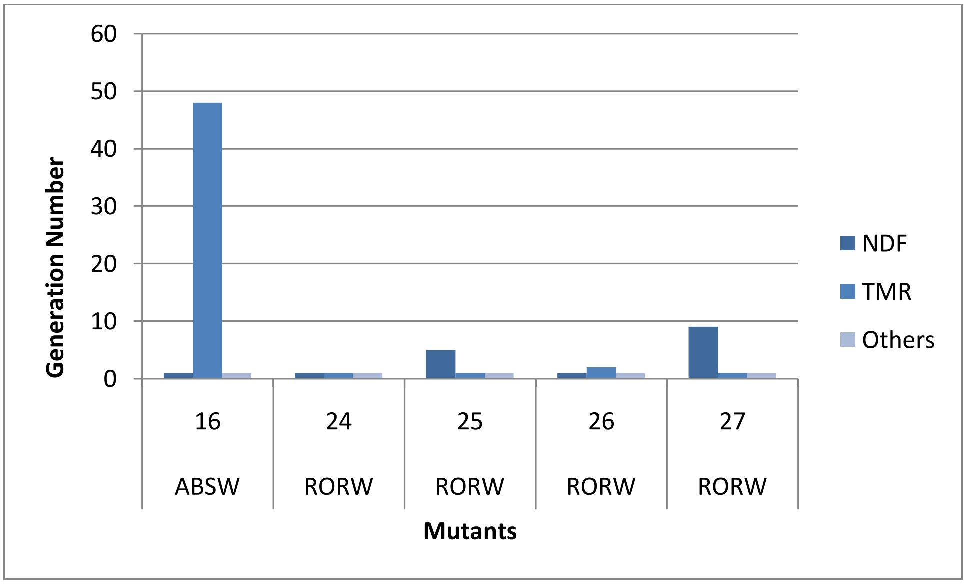

Figure 8 Group 1 Mutants Coverage in Query 3

Group Two Mutation Coverage

Table 20 contains full description for each Mutant that have invisible path in the mutants that related to Query 3.

Table 4 Group 2 Mutants Coverage in Query 3

Mutants	Invisible Paths	Reasons
M13	NDF	There are many values for

		Location_id in Location table.
M14	TMR	In department table department_id is PK and Location_id have value for each record in table.
M15	NDF	Department_id PK and property to become null is 0 and Location_id have value for each record in table
M17,M20	TMR	Department_id is PK and location_id always positive value.
M21	TMR	Department_id is PK and manager_id in Location table have unique value which not duplicated.
M22	TMR	Department_id is PK
M23	NDF	Location_id in Location table have many value and condition is opposites.
M28	NDF	Condition always true.
M29	TMR	Department_id is PK
M30	TMR	Department_id is PK and different range of data for columns in query that replaced.
M31	TMR	Department_id is PK and

		different range of data for columns in query that replaced.
M33	TMR	Department_id is PK
M37	TMR	Department_id is PK

Query 4

SELECT e.employee_id,j.max_salary from employees e, jobs j where e.job_id=j.job_id and e.commission_pct<=?xd?

Coverage for Three Types of Exceptions

Figure 9 shows that NDF have high ratio next to others exception. But TMR have lowest percentage which mean that TMR have invisible paths because mutants that generated for Query 4 replace many columns with other in different column that make ability to appear data in many row is rarely.

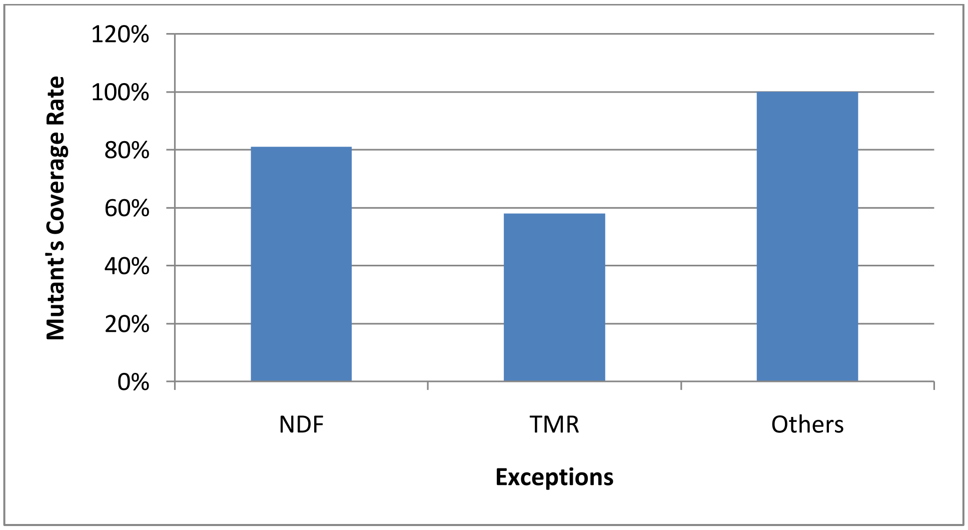

Figure 9 Total Mutants Exceptions Coverage in Query 4

Group One Mutation Coverage

Figure 10 shows that all exceptions coverage early. It appears all exceptions covered at first generation from GA. The worst case that in second generation which means good result.

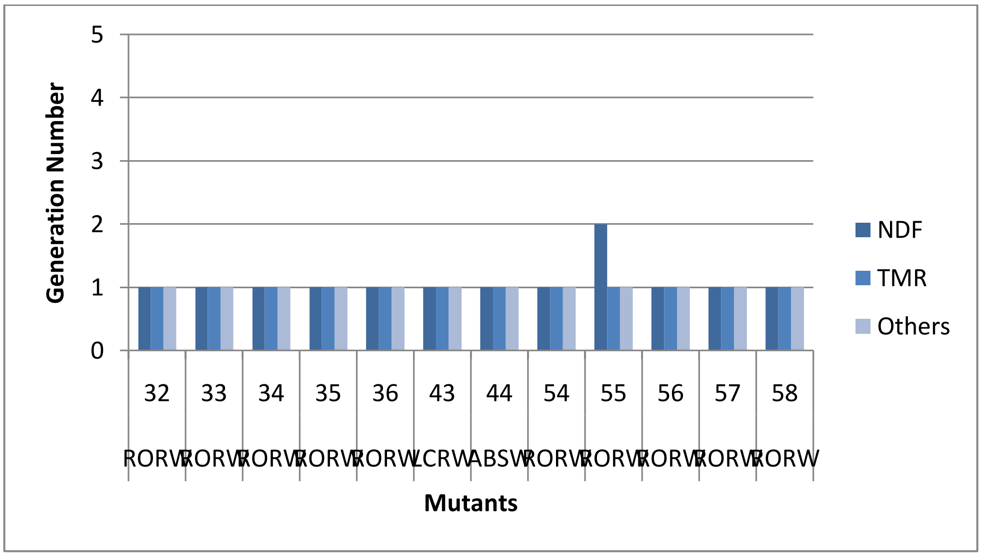

Figure 10 Group 1 Mutants Coverage in Query 4

Group Two Mutation Coverage

Table 21 describes the reasons of invisible paths that appeared in some case.

Table 5 Group 2 Mutants Coverage in Query 4

Mutants	Invisible Paths	Reasons
M23,M24	NDF	Commission_pct have many null values in employees table that make condition always true.

M27,M28,M29,M30	TMR	First_name,last_name,email,phone_number have different value when replacing with job_id and different with datatype.
M37	TMR	Condition(1=0) always false
M38	TMR	Job_id different value and datatype with job_title
M39	NDF	Values generated always true in condition of mutant.
M45,M48	NDF	Commission_pct always positive values.
M46,M47	TMR	Values generated may be come true but additional or sub values (1) to condition make condition false.
M49,M50,M51,M52 ,M53	TMR	The values not covered all many rows that generated by GA
M55	NDF	Commission_pct have values in table or null so the condition cant covered as true value.

Query 5

Coverage for Three Types of Exceptions

Figure 11 shows that TMR exceptions ratio high in the meanwhile NDF have low ratio. NDF have many invisible paths, which related to the nature of query that executed.

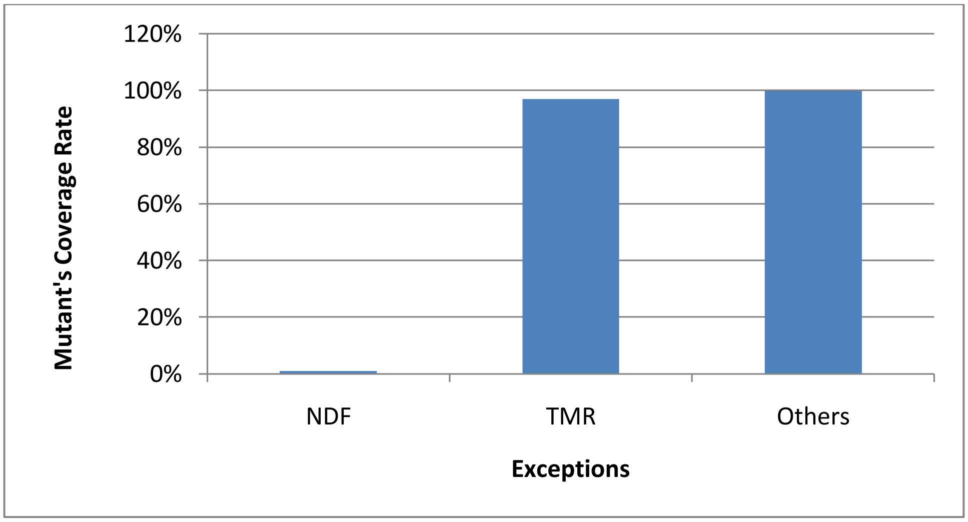

Figure 11 Total Mutants Exceptions Coverage in Query 5

Group Two Mutation Coverage

All mutants that related to query 5 have invisible path except M8 that may generate value to cover all exceptions.

Query 6

select job_id,avg(salary) from employees group by job_id having sum(salary)<?xi?

Coverage for Three Types of Exceptions

Figure 12 shows that TMR and Others exceptions have high percentage with 100%. but NDF exceptions have less percentage than other. The invisible path in these exceptions makes this low percentage.

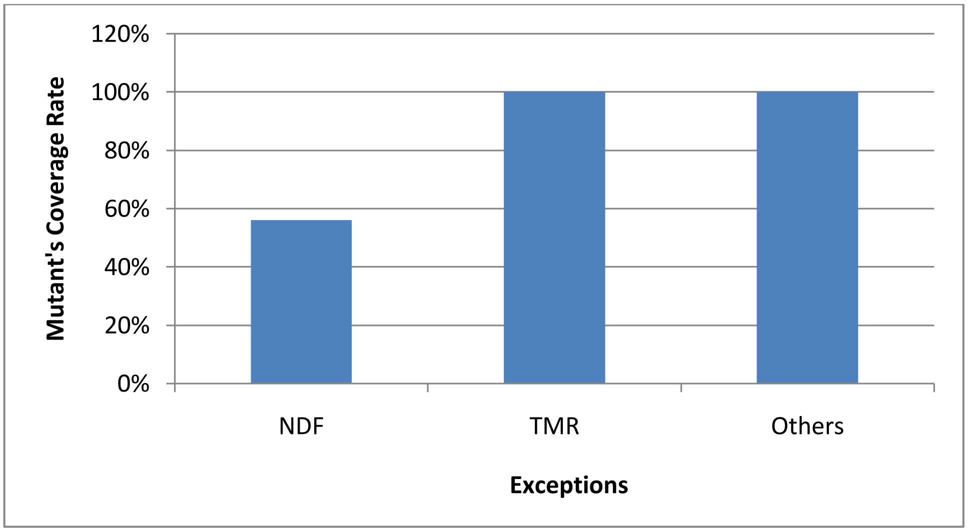

Figure 12 Total Mutants Exceptions Coverage in Query 6

Group One Mutation Coverage

As show in Figure 13 all exceptions covered early form first generation. But in some case like M45 NDF appear later that because there is invisible path to cover it when using < operation. In M47 the TMR exceptions appear later when the generation number become later to cover exceptions.

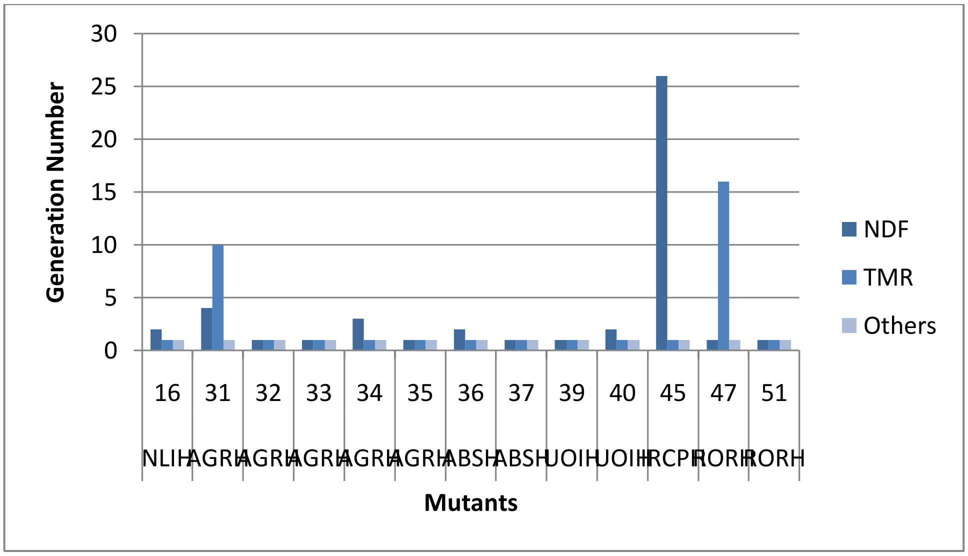

Figure 13 Group 1 Mutants Coverage in Query 6

Group Two Mutation Coverage

Table 22 shows why each mutant in Query 6 has invisible path.

Table 6 Group 2 Mutants Coverage in Query 6

Mutants	Invisible Paths	Reasons
M27,M48,M49,M50	NDF	Salary column have many values and sum (salary) have many values.
M38,M41	NDF	Salary has positive value always.

Query 7

select job_id from job_history where end_date between ?xu? and ?yu?

Coverage for Three Types of Exceptions

All exception coverage with high ratio as shown in Figure 14. That mean no invisible path existed.

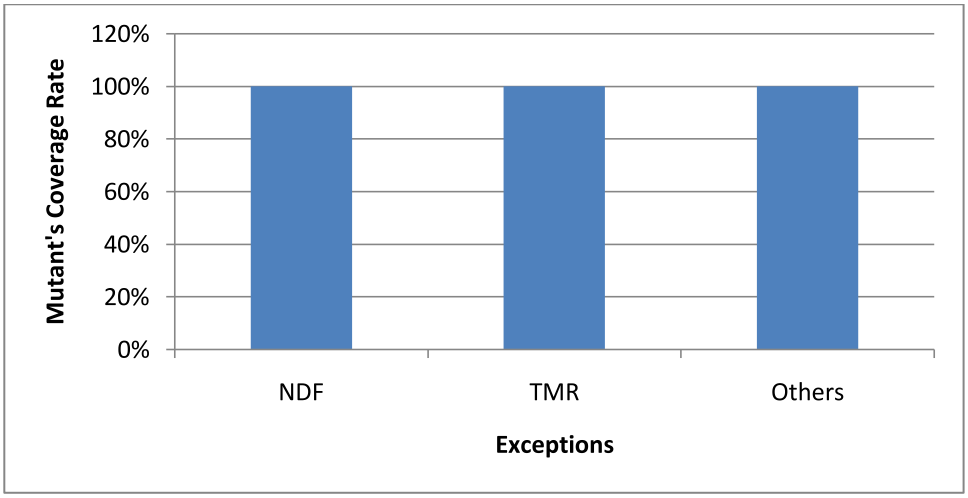

Figure 14 Total Mutants Exceptions Coverage in Query 7

Group One Mutation Coverage

All exceptions coved early in mutants in query 7 as show in Figure 15. In M5 NDF covered later because the generations of date depend of range that has minimum and maximum of values is generated rarely.

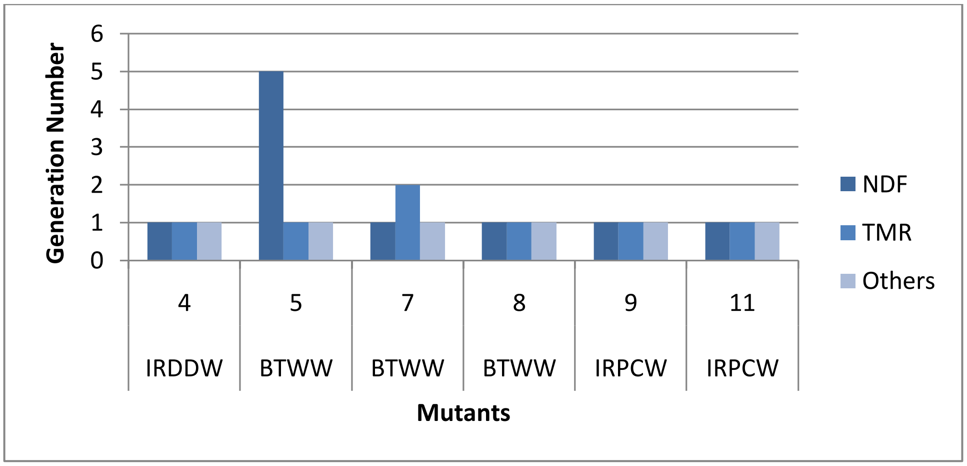

Figure 15 Group 1 Mutants Coverage in Query 7

Query 8

SELECT region_name from countries c, regions r where
r.region_id=c.region_id and
c.country_id=?xc?

Coverage for Three Types of Exceptions

In mutants of query 8 each of NDF and Others exceptions covered with high percentage as Figure 16 shows. But there are some invisible paths in mutants that make TMR exception appears with low ratio.

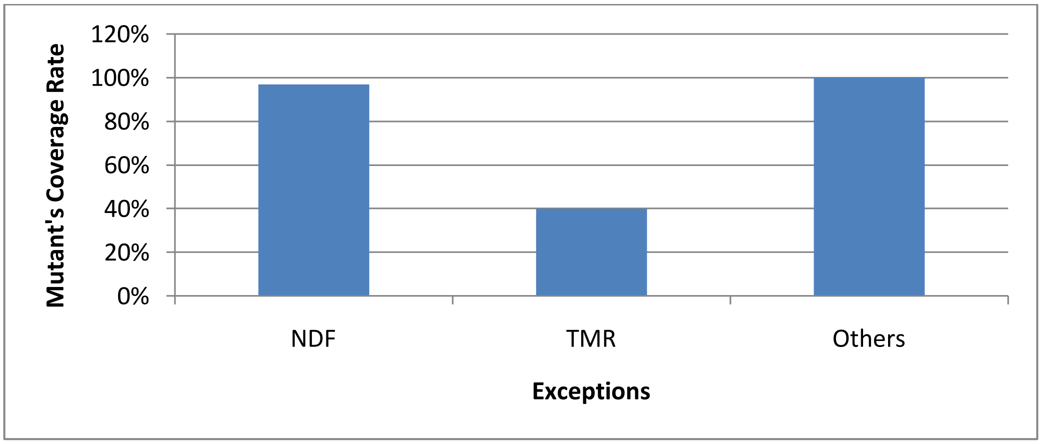

Figure 16 Total Mutants Exceptions Coverage in Query 8

Group One Mutation Coverage

Figure 17 shows NDF and Others exceptions covered from first generation for each mutants of query 8. But in seven mutants the invisible paths make TMR exceptions covered later with 48 generations. But when we look at table 23 which show the experiments results for query 8, the mutants that covered group 3 is less than that covered group 1 that mean the testing is get the best result.

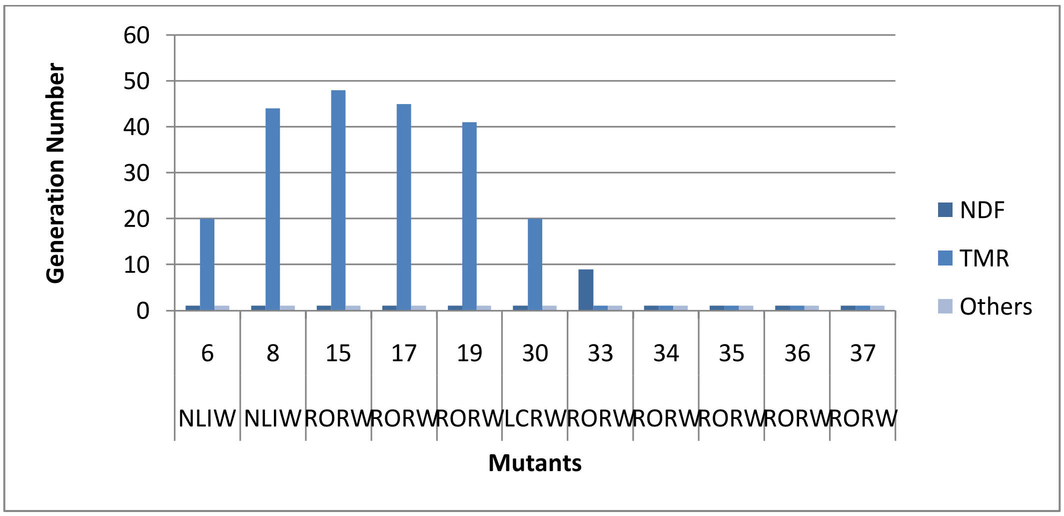

Figure 17 Group 1 Mutants Coverage in Query 8

Group Two Mutation Coverage

Table 23 shows that many mutants in Query 8 have invisible paths. Each mutant in table show the reason of invisible path that make mutant related to Group 2.

Table 7 Group 2 Mutants Coverage in Query 8

Mutants	**Invisible Paths**	**Reasons**
M5,M9,M11,M12, M21,M7,M23,M24,M32	TMR	In HR schema every country_id or country_name related to only one region_name
M10,M13,M22,M25	TMR	Region_id has positive value
M20	TMR	Condition (1=0) always false
M26	NDF	Always condition have a values
M33	NDF	Condition Country_id<>values covered because table contain many values for country_id

Query 9

Coverage for Three Types of Exceptions

All types of exceptions covered with high percentage as shown in Figure 18 for query 9. That means there are no invisible paths in these query mutants.

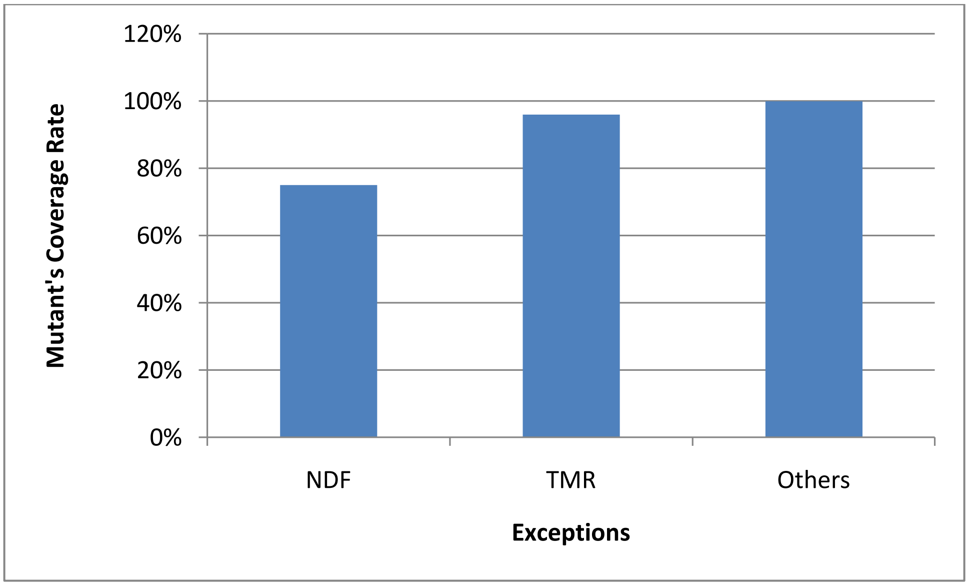

Figure 18 Total Mutants Exceptions Coverage in Query 9

Group One Mutation Coverage

Figure 19 shows that all types of exceptions covered early for large number of mutant's related to query 9, except M46 and M49 that covered TMR and Others exceptions covered early and NDF covered later because the value generated by GA found later. This value depends on the values in HR schema that haven't diversity values for salary column.

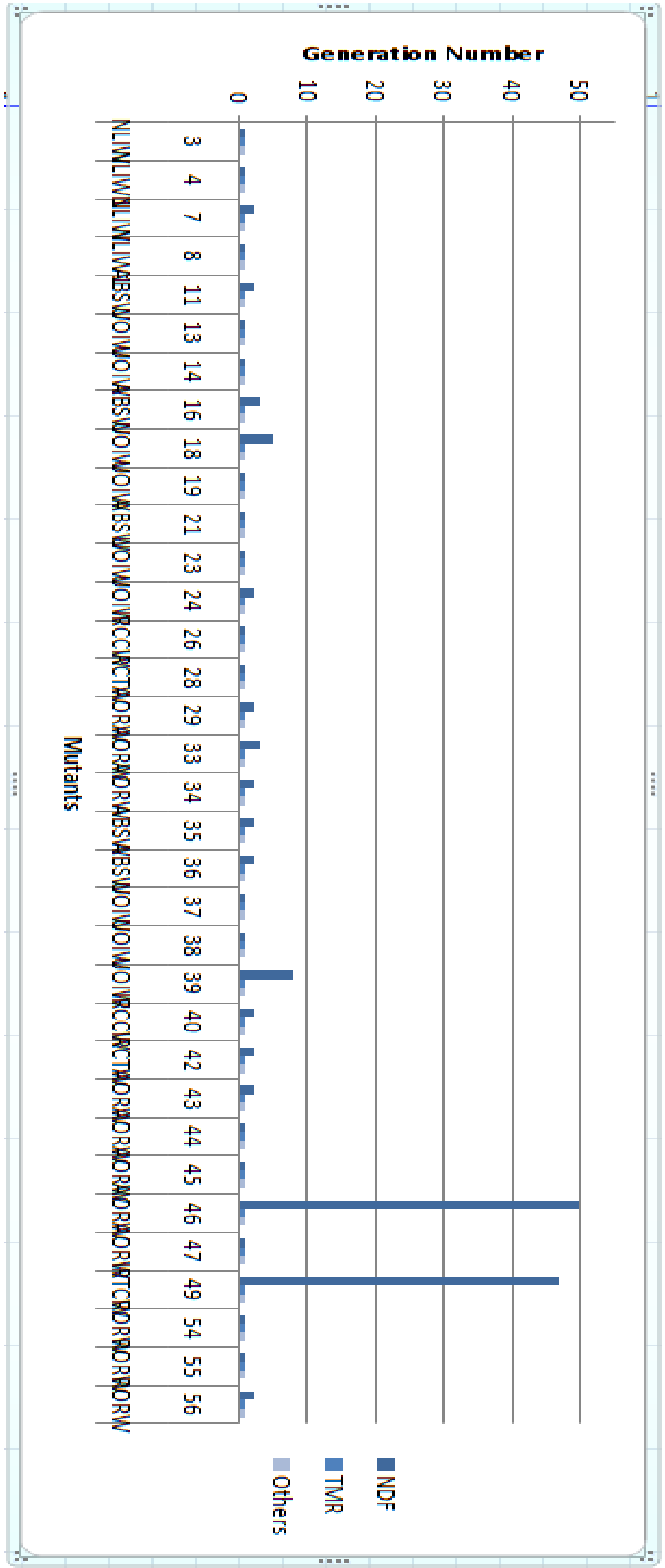

Figure 19 Group 1 Mutants Coverage in Query 9

Group Two Mutation Coverage

Many mutants in Query 9 related to Group 2 as shown in Query 9. There is full description in table 24 for reasons of invisible paths of query 9.

Table 8 Group 2 Mutants Coverage in Query 9

Mutants	Invisible Paths	Reasons
M12,M15,M17,M20 M22,M25	NDF	Columns in condition have always positive value
M30,M31,M32,M46	TMR	Range of data for the column in condition and generated by GA vary small comparing with the condition.
M53	NDF	Columns in condition have many values. Opposite with condition columns<>values.

Query 10

Coverage for Three Types of Exceptions

There are number of invisible paths that make NDF and TMR have low percentage for coverage as that appear in Figure 20. But others exception has 100% ratio coverage for query q10 and its mutants.

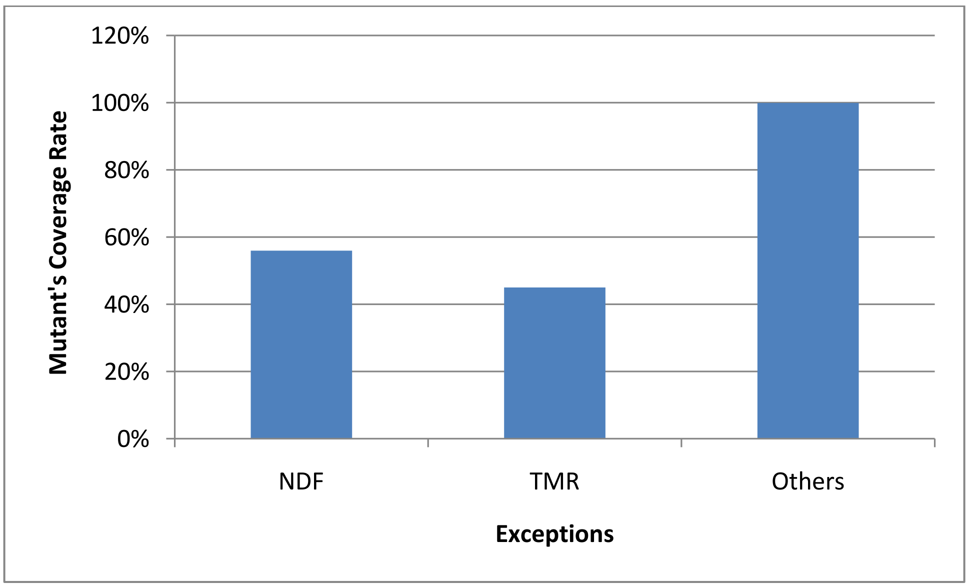

Figure 20 Total Mutants Exceptions Coverage in Query 10

Group Two Mutation Coverage

Table 25 shows six mutants of Query 6appear in Group 2 because having invisible path. Each mutant has reason of invisible path.

Table 9 Group 2 Mutants Coverage in Query 10

Mutants	Invisible Paths	Reasons
M8,M11	TMR	Columns in condition part have only positive values.
M15	NDF	Numbers generated by GA in condition not covered all values.
M16,M19,M22	NDF	The condition covered always.

Query 11

Coverage for Three Types of Exceptions

Two exceptions that NDF and Others covered with high ratio as shown in Figure 21. This percentage mean there is some exceptions like TMR has invisible paths that make ratio around 60%.

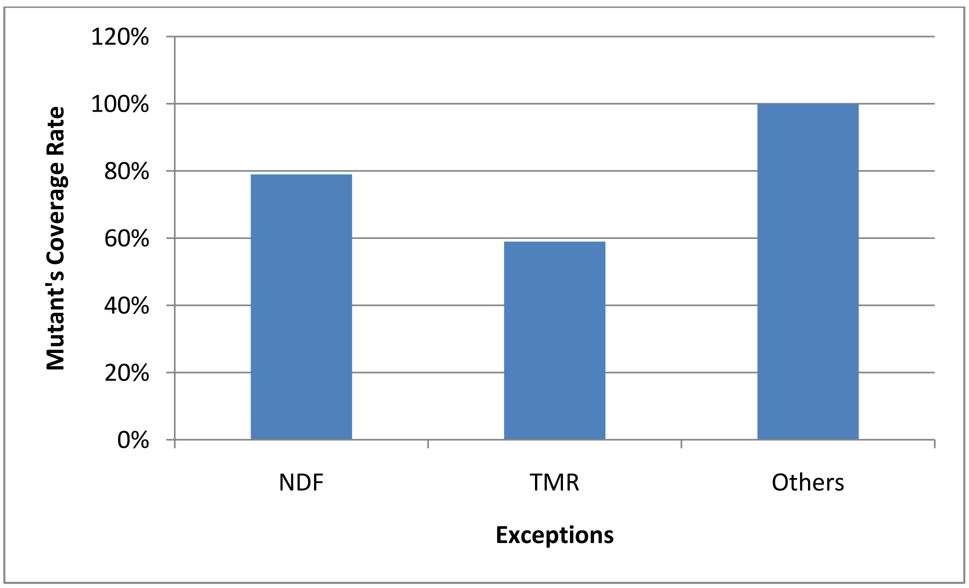

Figure 21 Total Mutants Exceptions Coverage in Query 11

Group One Mutation Coverage

NDF, TMR and others exceptions covered for all mutants related to query 11 from first generation. But in six mutants TMR exceptions covered later at generation number 43, which mean there is an invisible path for TMR exception.

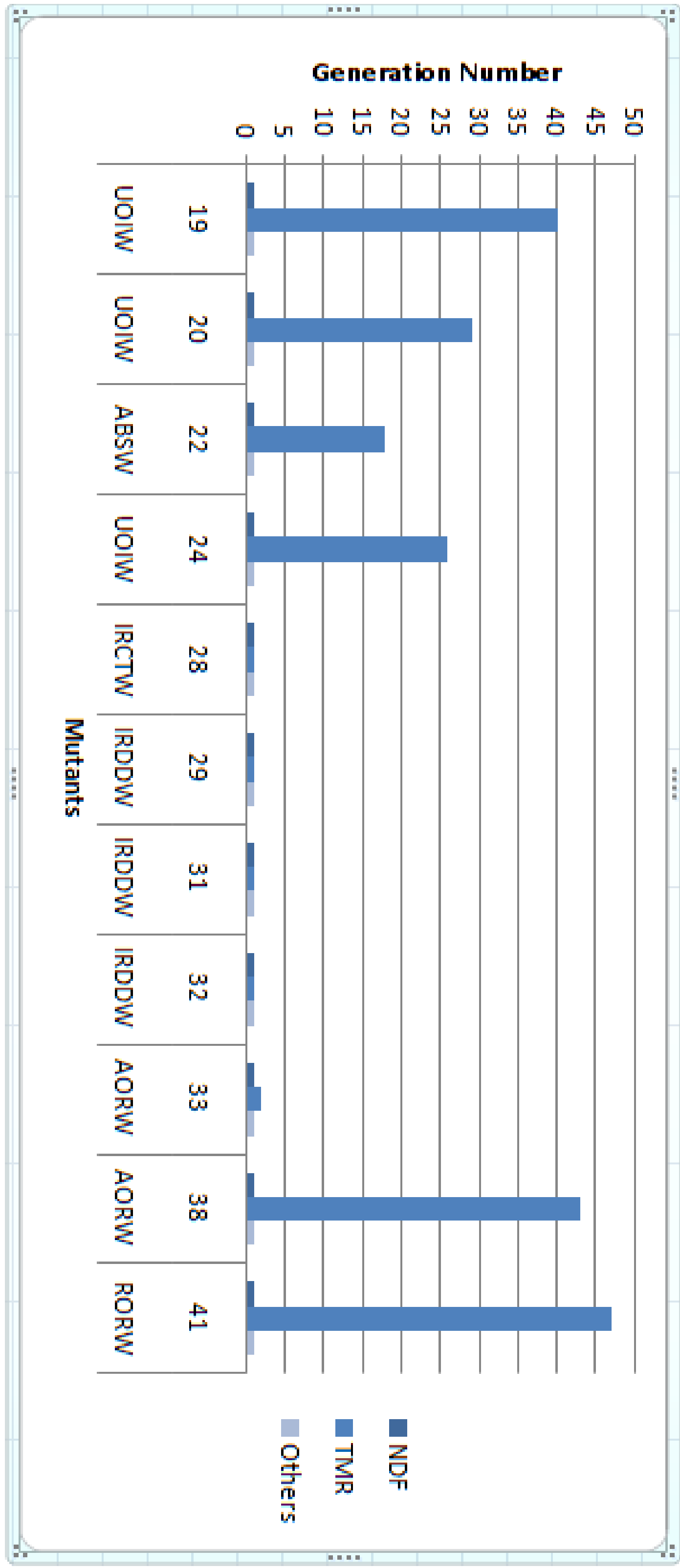

Figure 22 Group 1 Mutants Coverage in Query 11

Group Two Mutation Coverage

Table 26 shows that many mutants of Query 11 have invisible paths. So it appears in Group 2 and there are many reasons of these invisible paths and table describe it for each mutant**.**

Table 10 Group 2 Mutants Coverage in Query 11

Mutants	Invisible Paths	Reasons
M13,M14	NDF	Condition always true
M18,M21,M23,M26	TMR	Columns in condition have positive value only.
M25,M34,M35,M36, M37,M39	TMR	Values generated by GA become out of range for the columns in condition part.
M30, M25	NDF	Values generated by GA become out of range for the columns in condition part.
M42	NDF	Commission_pct has many different values

Query 12

Select location_id from locations where state_province is not null and street_address=?xc?

Coverage for Three Types of Exceptions

Two exceptions that NDF and Others covered with high ratio as shown in Figure 23. This percentage mean there is some exceptions like TMR has invisible paths that make ratio around 25%.

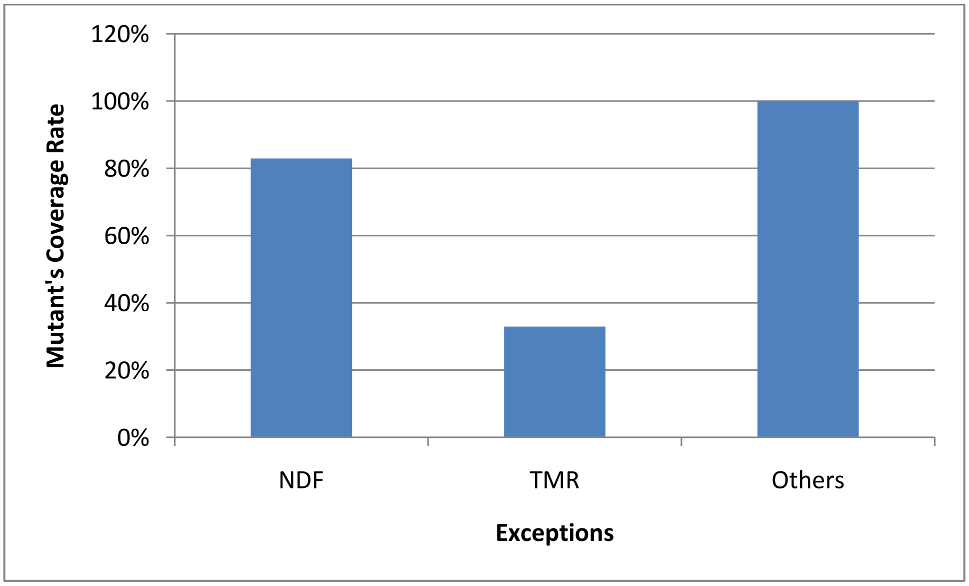

Figure 23 Total Mutants Exceptions Coverage in Query 12

Group One Mutation Coverage

All exceptions covered early in first generation for all mutants of query 12 that related to group 1.

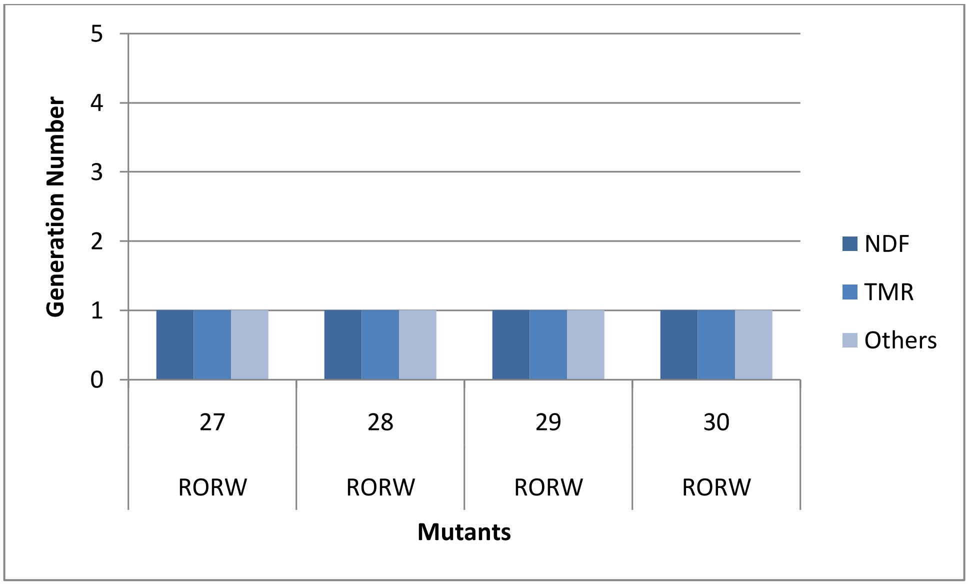

Figure 24 Group 1 Mutants Coverage in Query 12

Group Two Mutation Coverage

Table 27 shown mutants of Query 12 that related to Group 2 because they having invisible paths. There are many reasons for mutants. But as shown in table there are many mutants have same reasons that related to nature of query and HR schema.

Table 11 Group 2 Mutants Coverage in Query 12

Mutants	Invisible Paths	Reasons
M11,M12,M14,M15 ,M16,M7,M21, M22,M23,M24	TMR	Column in condition part has unique value (street_address is related to one Location_id).

M8, M26	NDF	Street_address has many different values
M17	NDF	Condition always true.

Query 13

Select * from departments d, locations l where d.location_id=l.location_id and l.city like ?xc?

Coverage for Three Types of Exceptions

Figure 25 show that almost all exceptions have high percentage for coverage. TMR exception has lowest ratio with 60%.

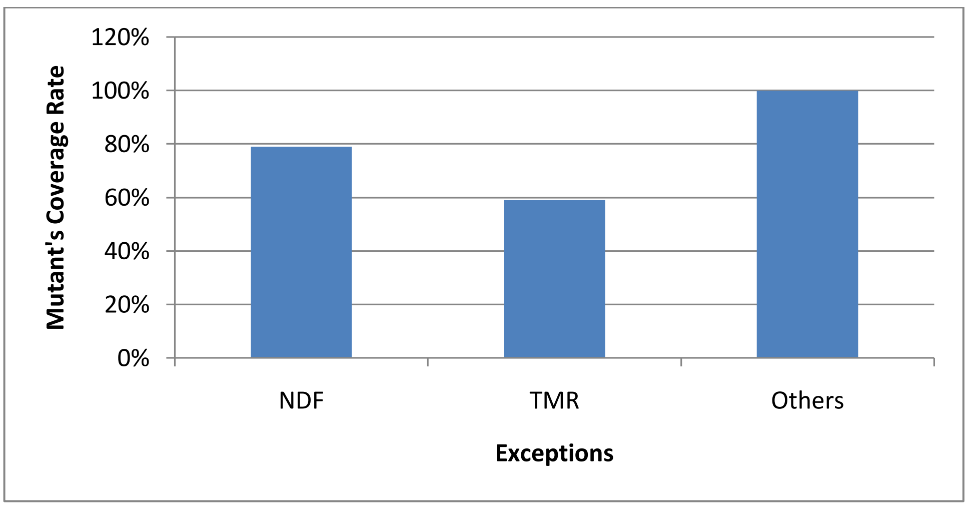

Figure 25 Total Mutants Exceptions Coverage in Query 13

Group One Mutation Coverage

NDF and other exceptions covered for all mutants appear in Figure 26 from first generation. But as this figure shows TMR exception need about 36

generation in GA to covered there are some investable paths that make this late of converge.

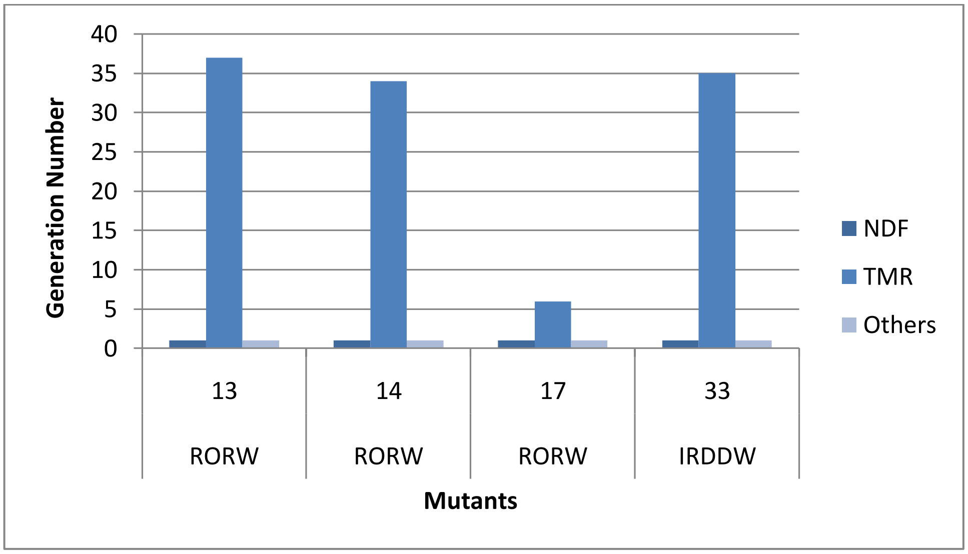

Figure 26 Group 1 Mutants Coverage in Query 13

Group Two Mutation Coverage

Table 28 shown many mutants related to Group 2 because they having invisible paths. And the reasons also appear in table for each mutant.

Table 12 Group 2 Mutants Coverage in Query 13

Mutants	Invisible Paths	Reasons
M4	TMR	Location_ID always have a value so condition always false. And TMR can't appear.
M7,M10,M21,	TMR	Location_ID has positive

M24		value.
M8,M9,M19, M22,M23	TMR	Condition can't cover any time. And always false
M11,M12	TMR	Condition can't cover any time because values for each column have different range of value.
M25	NDF	Always condition covered.

Query14

select job_title,max(salary) from employees e, jobs j where e.job_id=j.job_id group by job_title having max(salary)>?xi?

Coverage for Three Types of Exceptions

TMR has lowest percentage as shown in Figure 27 shows. That means it has invisible paths not like NDF and Others exception.

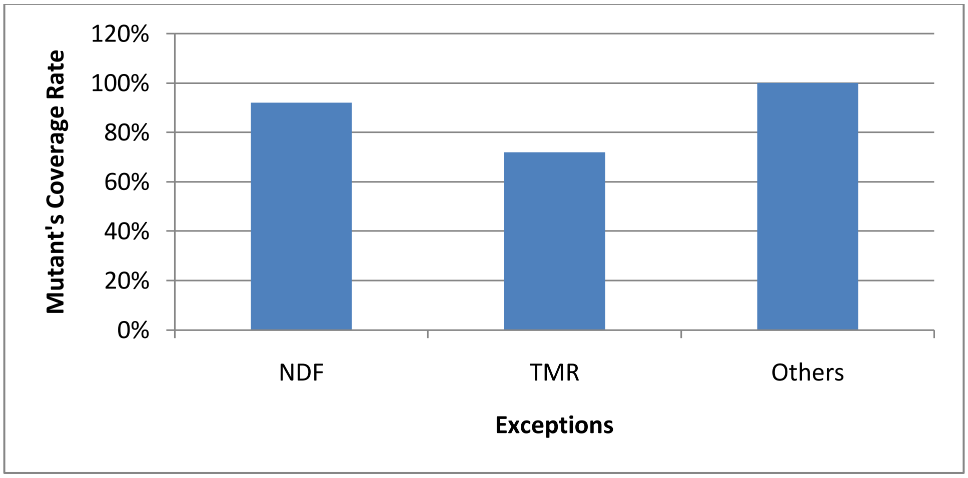

Figure 27 Total Mutants Exceptions Coverage in Query 14

Group One Mutation Coverage

All exceptions covered from first or in worst case in second generation as shows in Figure 28. But in M41, M54, M56 and M58 covered in later generation stage. The invisible paths appear because generation not covered condition clause and also it depending on range of HR data.

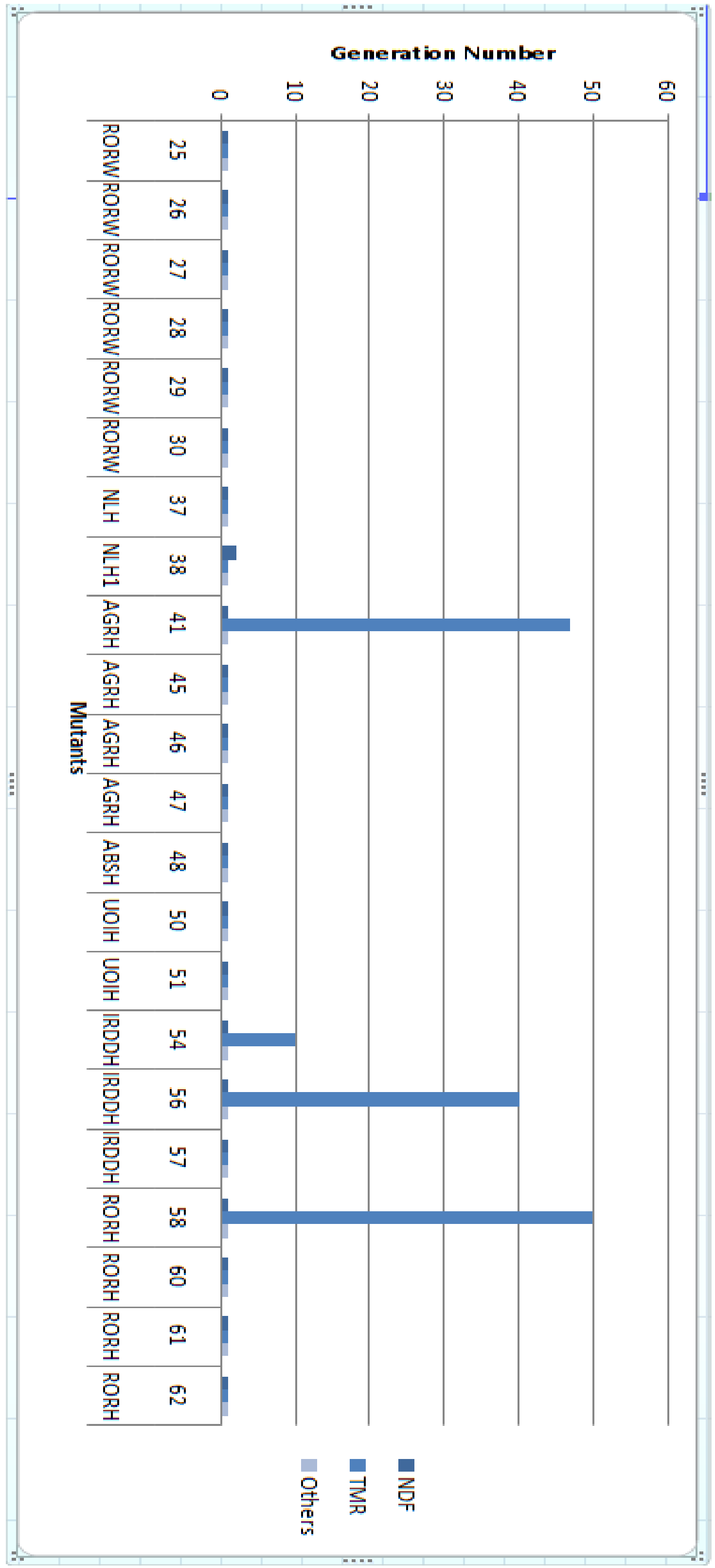

Figure 28 Group 1 Mutants Coverage in Query 14

Group Two Mutation Coverage

Table 29 shown numbers of mutants related to Query 14 belongs to Group 2 because they having invisible paths.

Table 13 Group 2 Mutants Coverage in Query 14

Mutants	Invisible Paths	Reasons
M20,M21,M22,M23, M24,M31,M32	TMR	Condition always false.
M43,M44,M55	NDF	Range of column in condition part not covered operation in condition.
M49,M52	TMR	Column in condition part has positive value.
M59	NDF	Always condition covered because column has many values.

Query 15

select count(*) from employees where department_id=?xi?

Coverage for Three Types of Exceptions

Figure 29 show that Other exception has 100% coverage percentage. But NDF and TMR not covered, each one have 0% ratio that related to query 15 that always related one record because the condition depend on count function that each time of execute return one value.

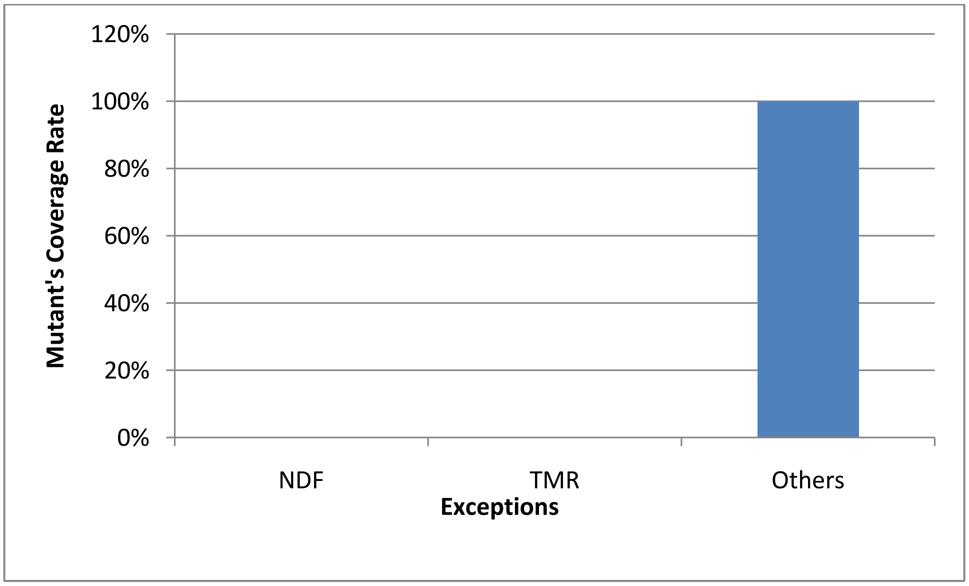

Figure 29 Total Mutants Exceptions Coverage in Query 15

Group Two Mutation Coverage

Query 15 always returns one value. so Group 2 covered by each mutants that generated by query 15 because function count that used in each mutants.

Query 16

select * from departments where manager_id is null and location_id=?xi?

Coverage for Three Types of Exceptions

NDF and Other exceptions covered with high percentage. But in TMR exception there is invisible paths that make ratio has lower.

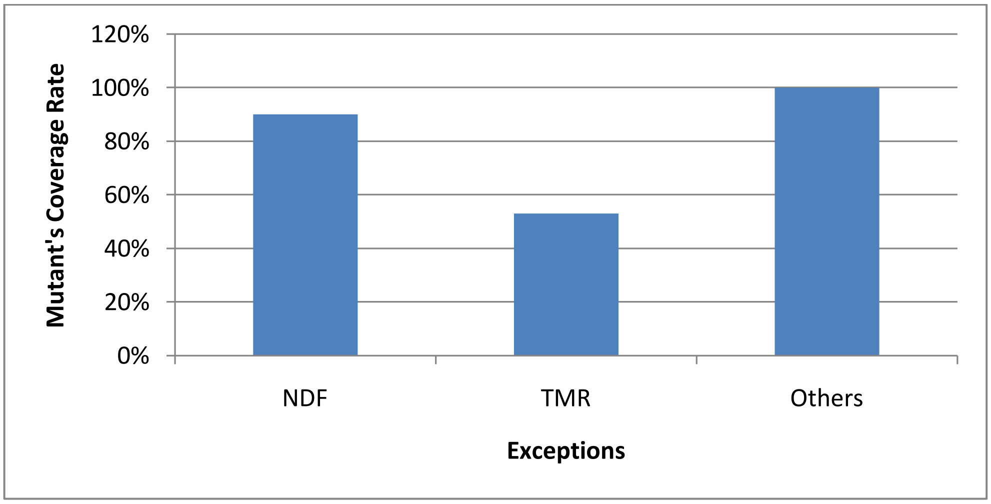

Figure 30 Total Mutants Exceptions Coverage in Query 16

Group One Mutation Coverage

All exceptions covered in most mutants from first generation. But in M6,M15 and M10 covered later as shown in Figure 31. And in M22 the NDF exception also covered in generation number 48. Because there is invisible paths in each exceptions.

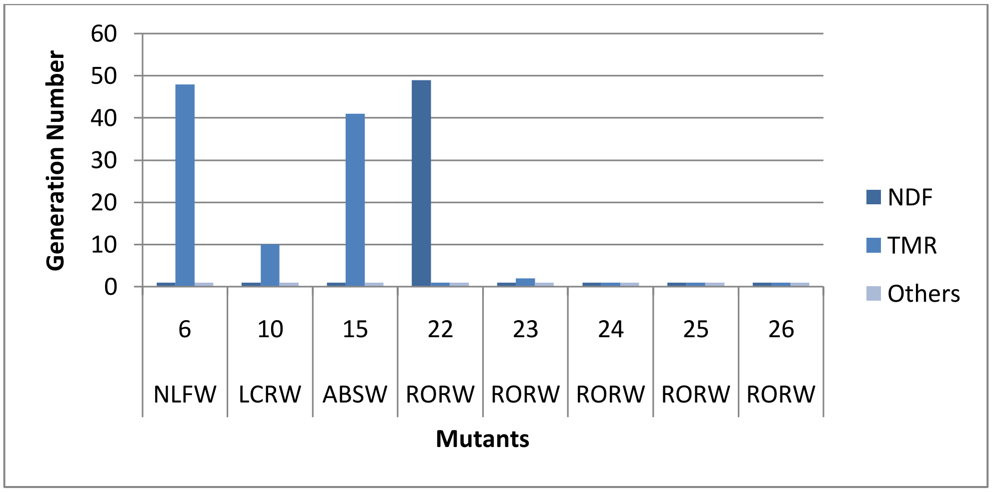

Figure 31 Group 1 Mutants Coverage in Query 16

Group Two Mutation Coverage

Table 30 have a reasons for each mutants having invisible paths and belongs to Group 2.

Table 14 Group 2 Mutants Coverage in Query 16

Mutants	Invisible Paths	Reasons
M3	NDF	Always condition covered because the column in condition part has many values.
M7,M9	TMR	Condition always false.
M10	NDF	Condition always true.
M16,M19	TMR	Column in condition part has

		positive value.
M20	TMR	Condition can't cover depending on database.
M21	TMR	Column in condition part can't duplicated because it is a PK.
M22	NDF	Column has many values.

Query 17

Coverage for Three Types of Exceptions

Figure 32 shows that NDF and Other exceptions covered with high percentage. But in TMR exception there is invisible paths that make ratio has lower value.

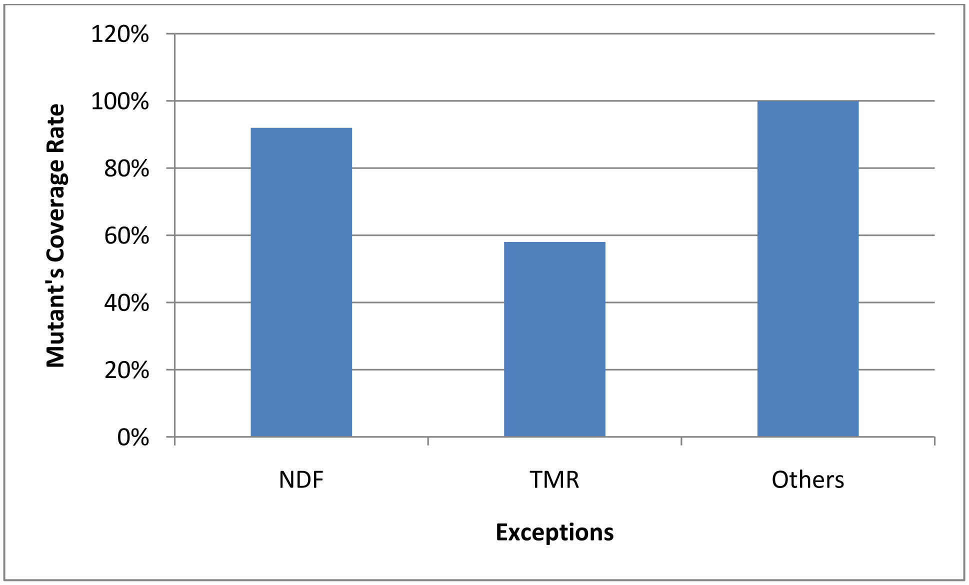

Figure 32 Total Mutants Exceptions Coverage in Query 17

Group One Mutation Coverage

In this query there are Varity of coverage for each mutant. As Figure 32 shows that in M17,M25,M27 and M28 the NDF exception covered later. In other hand in M18,M19,M20,M26,M35,M36,M39 and M40 TMR exception covered later. There are invisible paths that make coverage become later. But in other case all exceptions covered from first generation.

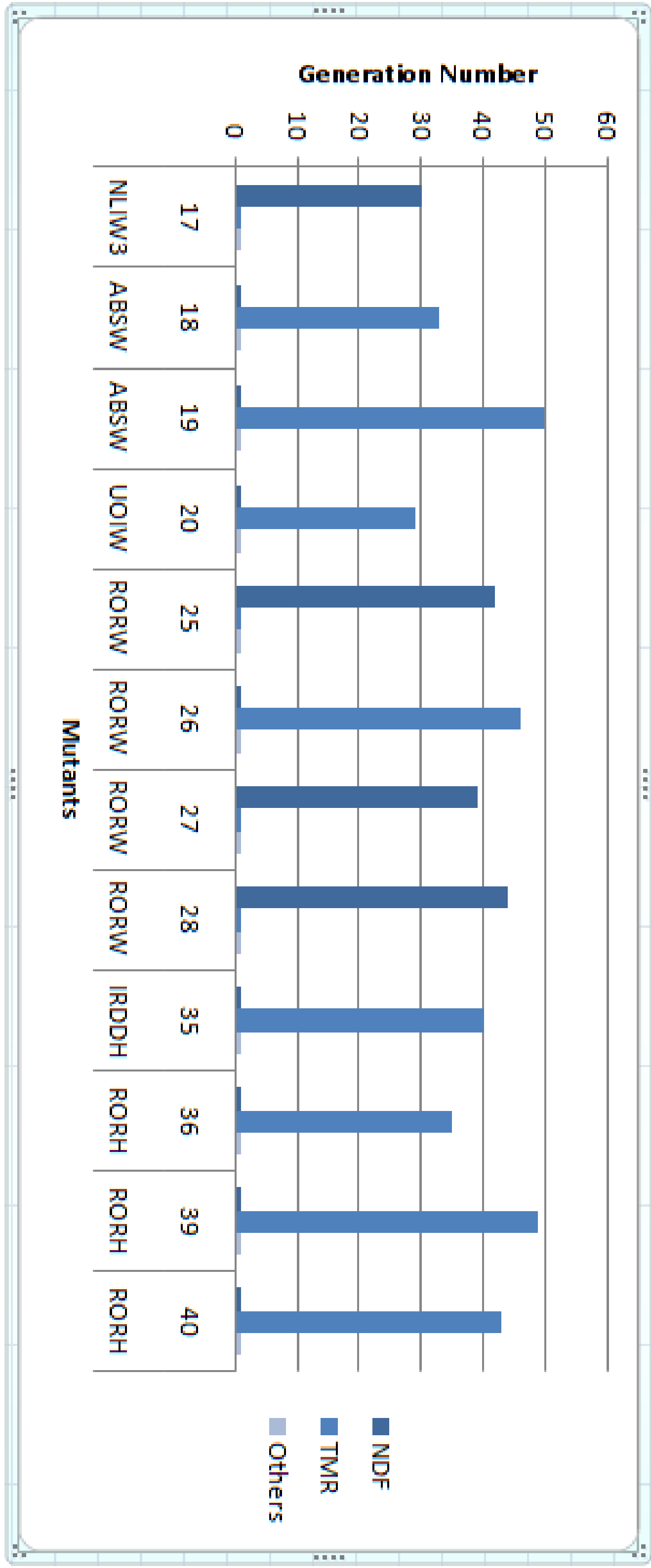

Figure 33 Group 1 Mutants Coverage in Query 17

Group Two Mutation Coverage

Table 31 contain seven mutants related to Query 17 that belongs to Group 2 because they having invisible paths.

Table 15 Group 2 Mutants Coverage in Query 17

Mutants	Invisible Paths	Reasons
M15	TMR	Column in condition part has always values.
M19,M22	TMR	Column in condition part always positive.
M23	TMR	Column in condition part related to one value.
M24	NDF	Region_id has many values.
M29,M41	TMR	Condition always false.

We test our system under HR schema in Oracle database. This schema is a small sample of data, so the results that will be better if we used schema contain large number of data and diversity in data. The HR schema not contains a Varity of data so there is low probability of generate value for any query by genetic algorithms i.e. SALARY column in EMPLOYEES table contains values like between 2100 – 24000. And the values increment by 100

(2100, 2200, 2400, etc.). The probability of generate values in each population in genetic algorithms end with (00) is rarely i.e. 2100 generating two zero in GA rarely to happen because it depends on individual that generate this new value.

Comparing results:

Previous sections showed the results of our experimental testing using new system. In this section we will compare these results (results from our new approach) with other approach that used same HR schema to test covering NDF, TMR and Others exceptions without fitness in function.

This comparison covered groups that appear in each query and percentage of covering for each one. In each stage we will describe the difference between two results.

Each query there is a table that contains the percentages of Mutants exceptions that covered in query for two approaches; GA without fitness and GA with fitness that used fitness in GA. Then two figures appear the generation number for coverage exceptions in each query. Group1 as described before shows the mutants that covered for each query, so each mutant for any query that belongs to group1 must appear in these two figures.

Each seventeen table below will have two columns. Column one contain percentage of coverage three types of exceptions for GA without fitness, and column two has a values for GA with fitness for all queries.

Tables in this chapter will described that GA with fitness for all queries has high percentages than GA without fitness, because in GA with fitness we use fitness value for each query that help us to know the range of data for each query for testing the coverage exceptions. This idea makes GA generate test cases with values depending on fitness value for queries.

Using Fitness values in GA with fitness for each query depend on the type of exception. In NDF we use the minimum and maximum values for columns that used in condition part in query. Then we enter these values (minimum and maximum) in GA part in our code to generate test cases and generate values for test cases. Knowing the fitness and range of data for any query helps GA with fitness to get best values than GA without fitness and this will appear below in the tables and figures result for all queries that used in our experiment. In TMR we used the count of values for the columns used in condition part of query. If the values that generated in GA for any test case appear more than one times the TMR exception coverage. Other cases covered

others exception that appear in GA with fitness with percentage 100% for each query.

The difference of percentage value for two approaches that will shows below related to using the fitness and range of data as we described before. Some queries have large different in percentage or in other hand have small difference in value, this Varity of difference depend on nature of query, this appears in many queries for example in query 15 there is no difference in percentage for each approach, because the nature of query is not have ability to get best values for any GA with fitness.

The difference in two approaches also will appear in figures below that have the generation number for mutants of queries that coverage exceptions and belongs to group 1. In GA without fitness some mutants in queries belongs to group 3 that need to test more than 50 times to coverage the exceptions, but in new approach some of these mutants covered early(less than or equal to 50 test case) and appear in group 1.

Query 1

Table 16 Percentages of Mutants Exceptions Coverage in Query 1

	GA without fitness	GA with fitness
NDF	92%	93%
TMR	49%	57%
Others	100%	100%

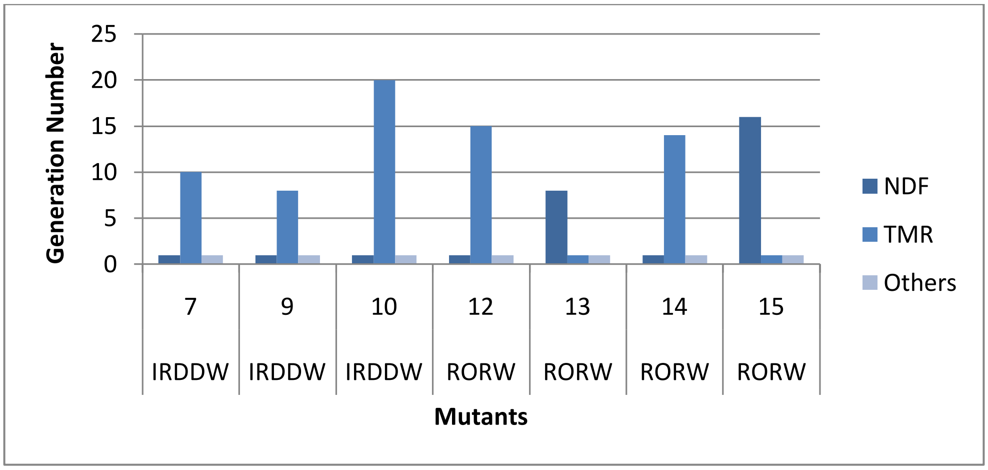

Figure 34 Group 1 Mutants Coverage in Query 1 for GA with fitness

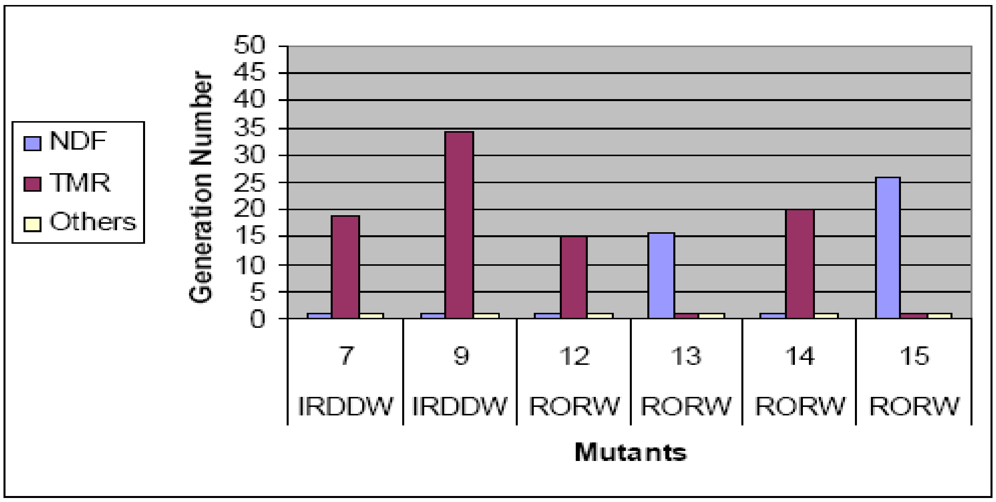

Figure 35 Group 1 Mutants Coverage in Query 1 for GA without fitness

The coverage percentage value in query 1 for NDF and others exception have convergent values for new and GA without fitness with best value for new approach. But in TMR exception there is 8% difference between two approaches 49% for old one and 57% for new one. fitness value that used in new approach make the range of data clear for GA that generate values in TCs early for each mutant in query 1.

Mutant 10 in query 1 appear in group 1 in new approach but in GA without fitness it appears in group 3 (need many test cases to appear). New approach idea helps us to appear some mutants appear early in less than 50 generation that make some mutants belongs to group 2 not in group 3.

Query 2

Table 17 Percentages of Mutants Exceptions Coverage in Query 2

	GA without fitness	GA with fitness
NDF	80%	80%
TMR	80%	80%
Others	100%	100%

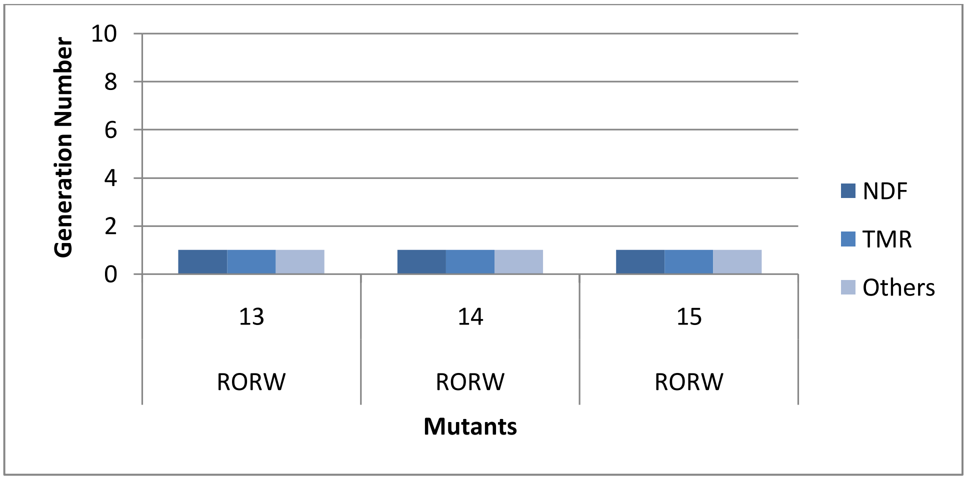

Figure 36 Group 1 Mutants Coverage in Query 2 for GA with fitness

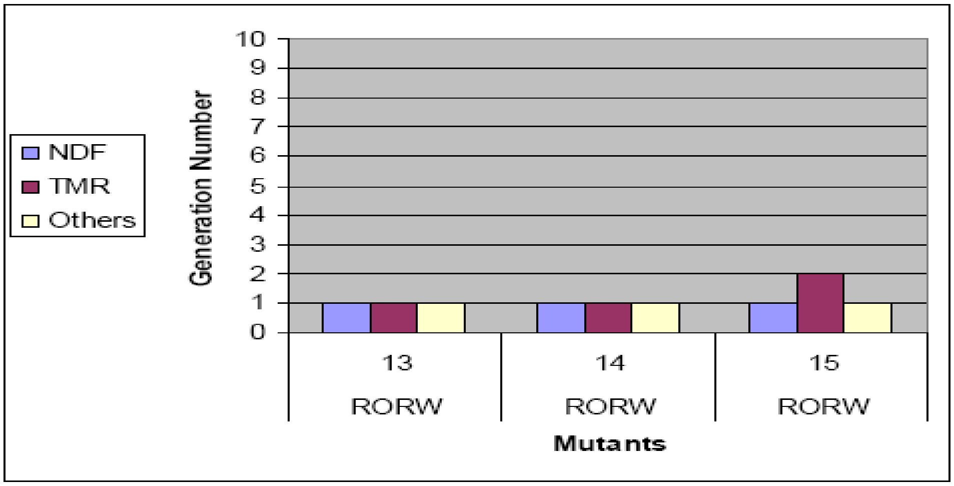

Figure 37 Group 1 Mutants Coverage in Query 2 for GA without fitness

The nature of query 2 make the results of two approaches has same percentage. No changes in result because all mutants generated for query 2 covered early as show in Figure 36 and Figure 37 from first generation with high personage coverage for all exceptions. There is small difference appear in these figures in mutant 15 with best value for new approach that covered early.

Query 3

Table 18 Percentages of Mutants Exceptions Coverage in Query 3

	GA without fitness	GA with fitness
NDF	81%	82%
TMR	35%	39%
Others	96%	100%

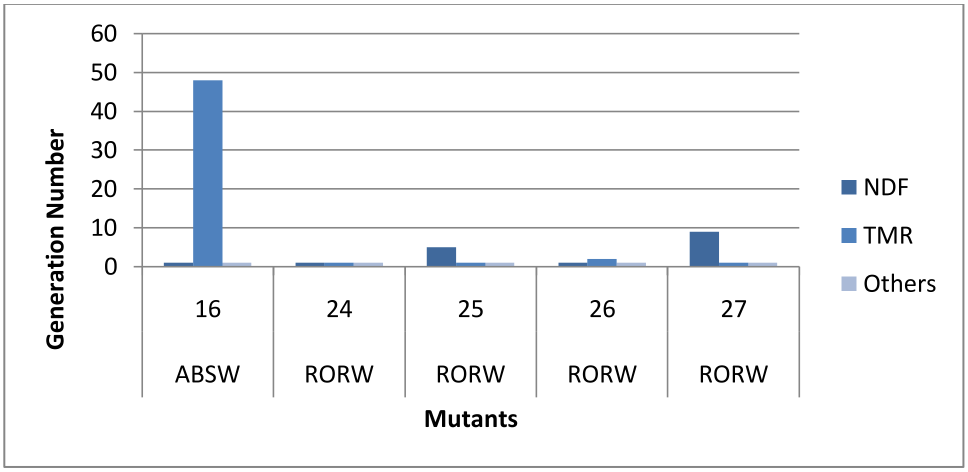

Figure 38 Group 1 Mutants Coverage in Query 3 for GA with fitness

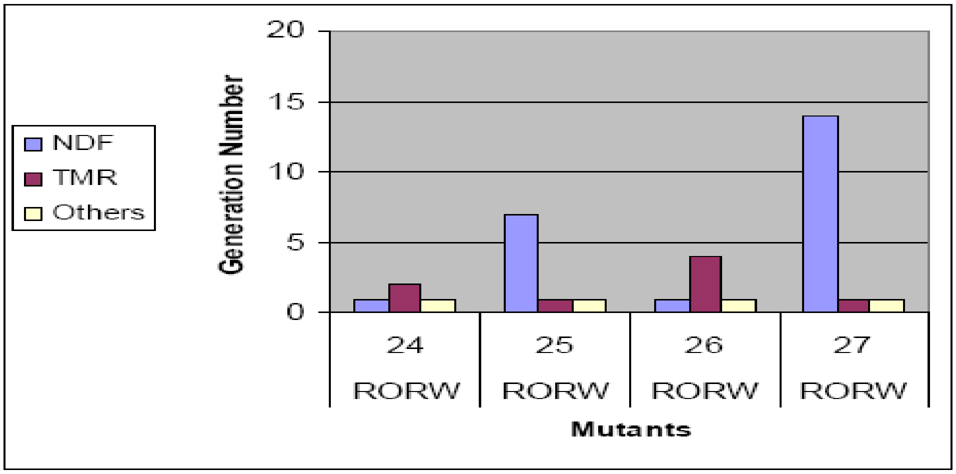

Figure 39 Group 1 Mutants Coverage in Query 3 for GA without fitness

New approach has best value that old one for query 3 as appear in figures and table above. 1% difference for NDF and 4% for TMR and 4% for Others exception.

Mutant 16 covered in new approaches in generation number 48 (belongs to group 1) but in GA without fitness this mutants appear in group 3 that need many TCs to covered exceptions.

Query 4

Table 19 Percentages of Mutants Exceptions Coverage in Query 4

	GA without fitness	GA with fitness
NDF	81%	81%
TMR	35%	58%
Others	95%	100%

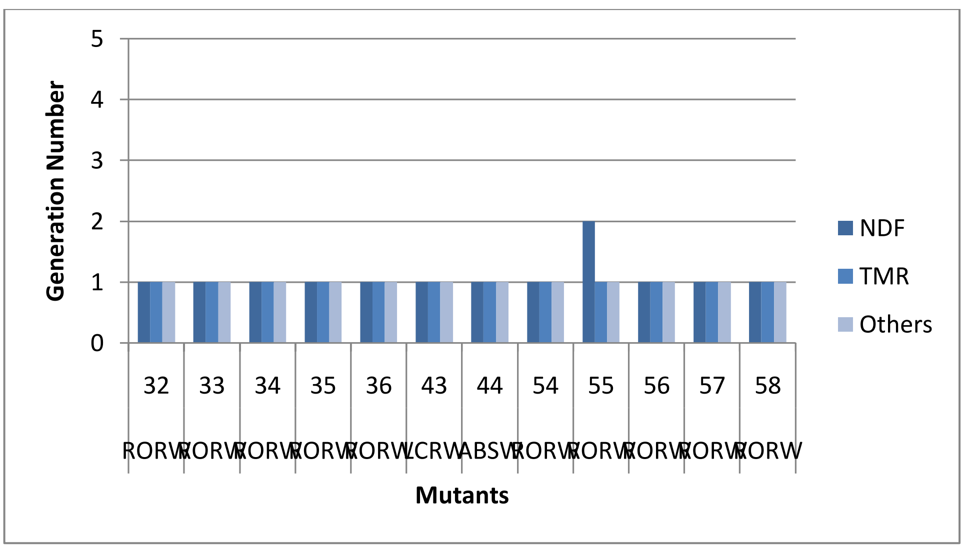

Figure 40 Group 1 Mutants Coverage in Query 4 for GA with fitness

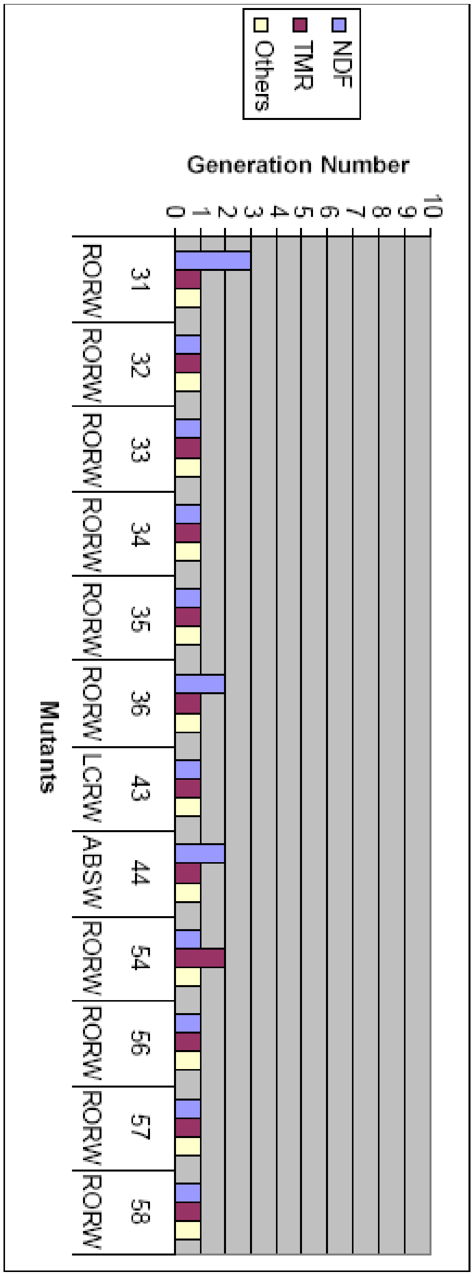

Figure 41 Group 1 Mutants Coverage in Query 4 for GA without Fitness

New approach have best values for all mutants related to query 4. All mutants covered in first generation except mutant 55 that covered in second generation which mean new approach have excellent values for coverage exceptions.

The difference between values for two approach show how new approach has good values, In TMR exception 23% and others exception 5%.

Query 5

Table 20 Percentages of Mutants Exceptions Coverage in Query 5

	GA without fitness	**GA with fitness**
NDF	0%	0%
TMR	98%	98%
Others	100%	100%

In query 5 no change appears in the results of two approaches. That related to nature of query and data in schema that make the coverage exception in this schema.

Query 6

Table 21 Percentages of Mutants Exceptions Coverage in Query 6

	GA without fitness	**GA with fitness**
NDF	51%	56%
TMR	100%	100%
Others	100%	100%

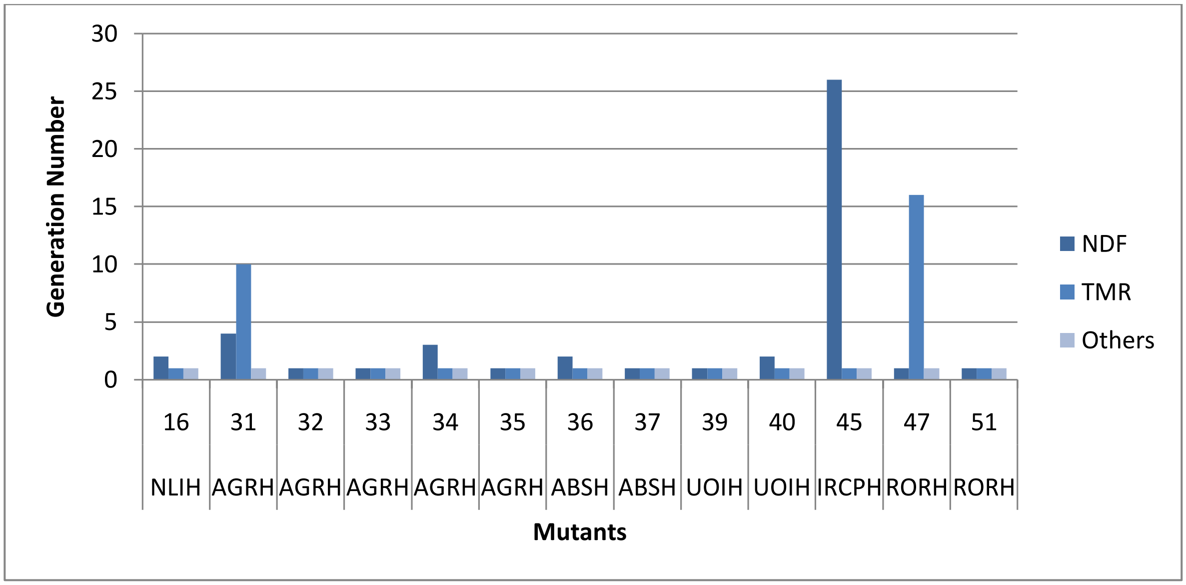

Figure 42 Group 1 Mutants Coverage in Query 6 for GA with fitness

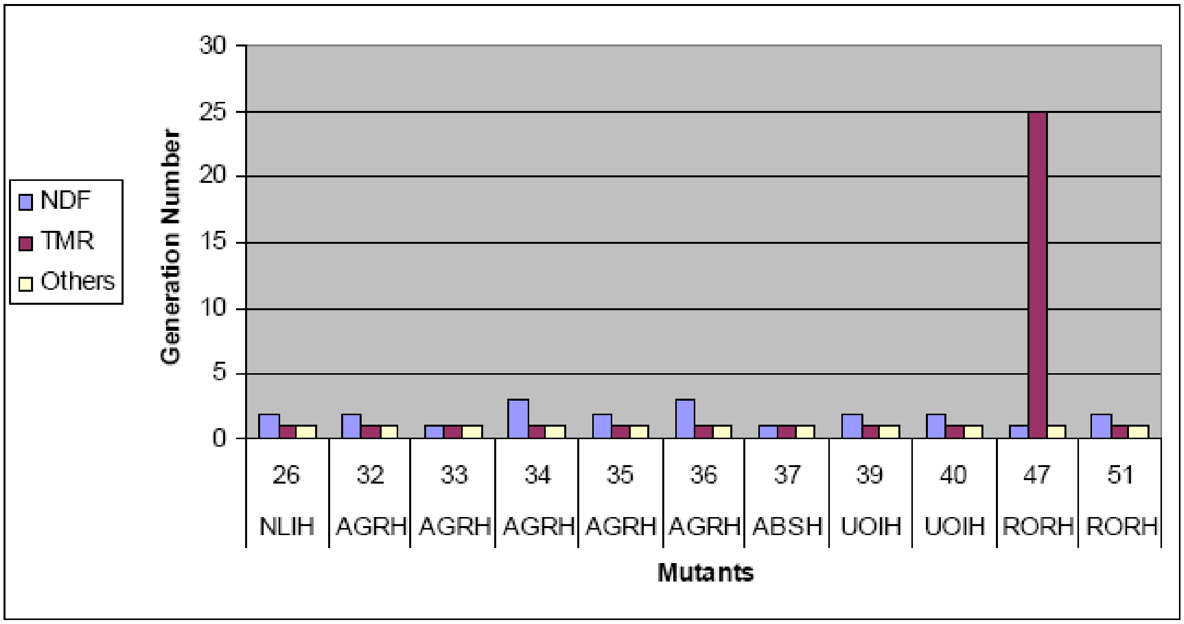

Figure 43 Group 1 Mutants Coverage in Query 6 for GA without Fitness

TMR and others exception covered in query 6 with high percentage for two approaches (100%). But there is little change appears in coverage percentage for NDF exception (5%).

The difference of the percentage in NDF exception occurred because some mutant appear in group 1 in new approach like mutant 31 and mutant 16, and increase the percentage of coverage NDF exception. These two mutants related to group 3 in GA without fitness (cowage exception in generations more than 50 times).

Query 7

Table 22 Percentages of Mutants Exceptions Coverage in Query 7

	GA without fitness	GA with fitness
NDF	100%	100%
TMR	100%	100%
Others	100%	100%

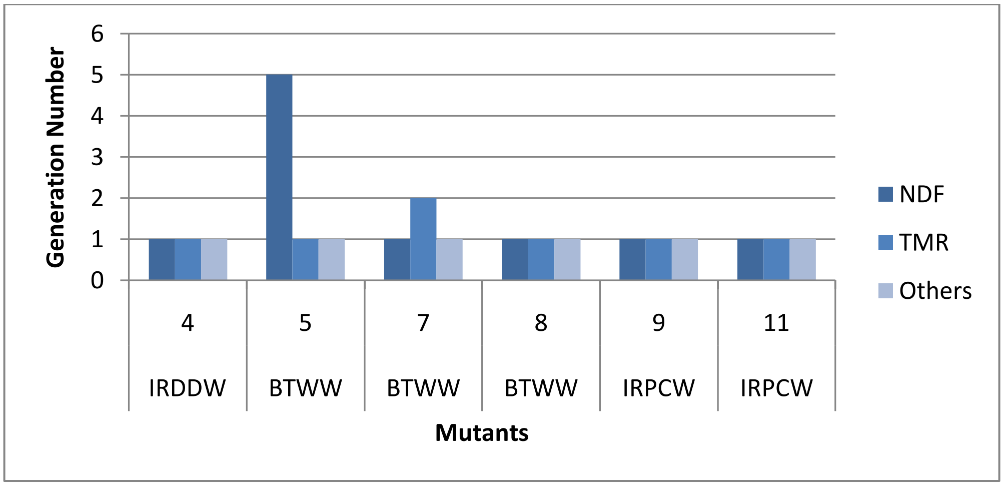

Figure 44 Group 1 Mutants Coverage in Query 7 for GA with fitness

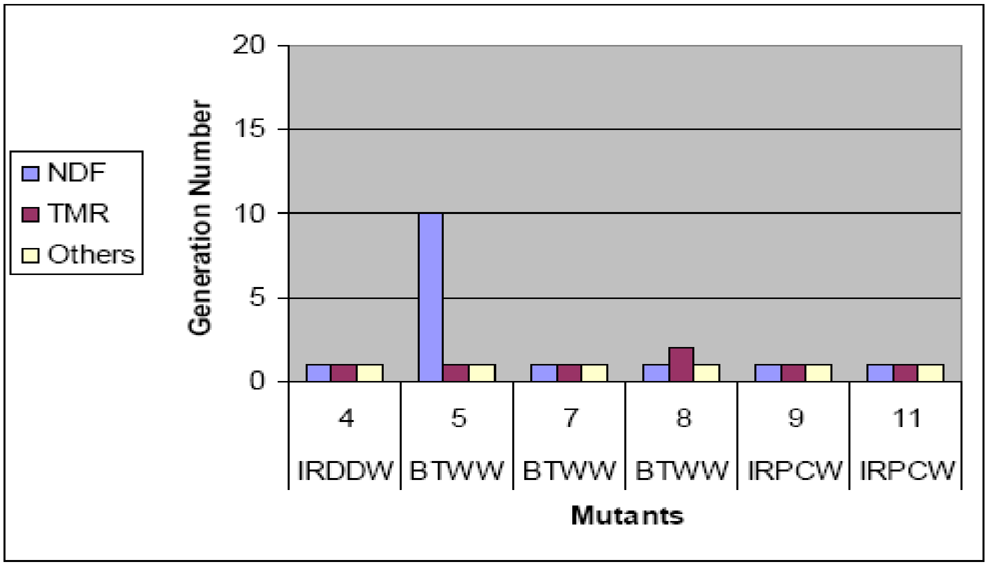

Figure 45 Group 1 Mutants Coverage in Query 7 for GA without Fitness

All exception in query 7 covered with high percentage values for two approaches with 100% ratio for all exception in two approaches.

The difference in values for two approaches is the coverage for mutants occurred early in new approach i.e. mutant 5 covered in generation number 5 but in GA without fitness covered in 10 times of generation.

Query 8

Table 23 Percentages of Mutants Exceptions Coverage in Query 8

	GA without fitness	GA with fitness
NDF	94%	97%
TMR	27%	40%
Others	97%	100%

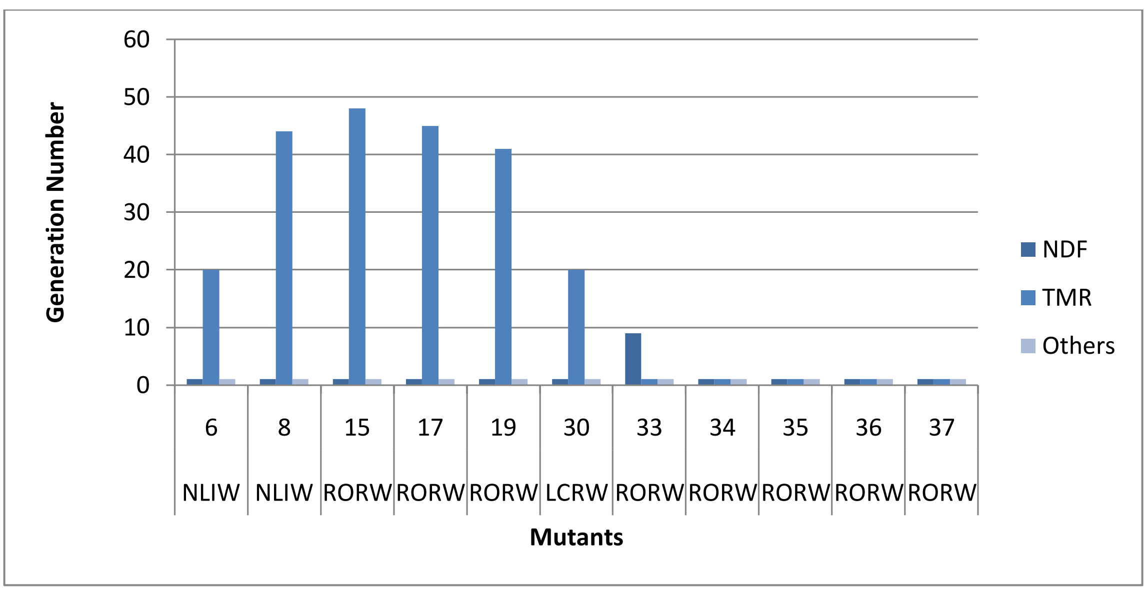

Figure 46 Group 1 Mutants Coverage in Query 8 for GA with fitness

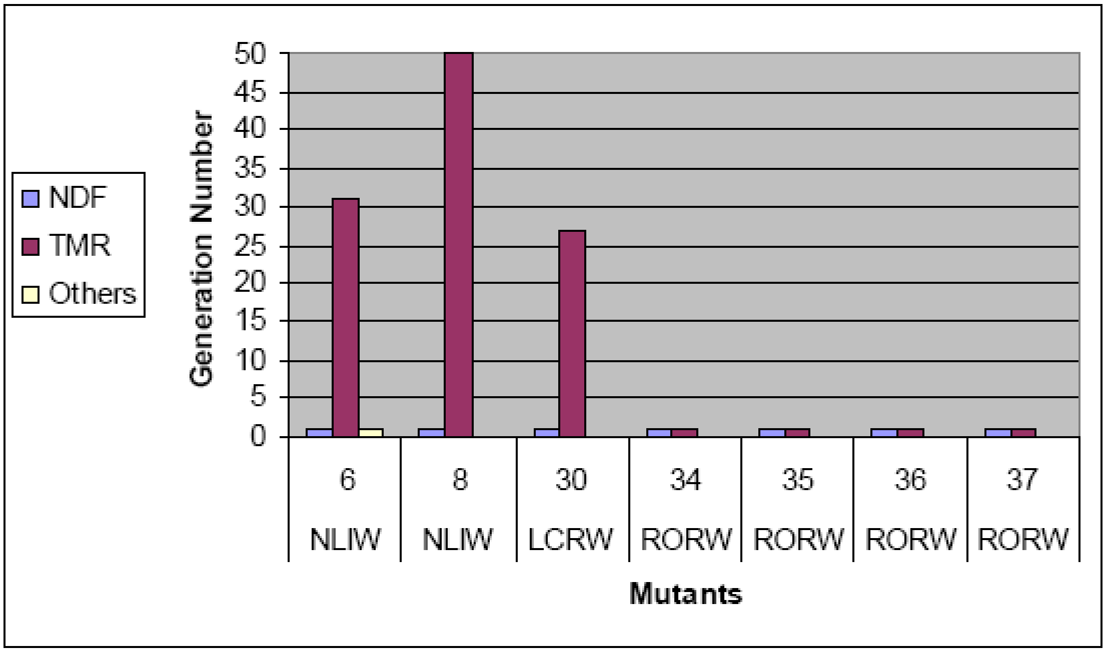

Figure 47 Group 1 Mutants Coverage in Query 8 for GA without Fitness

Four mutants appear in group 1 when we use new approach (M19, M17, M15 and M33). In GA without fitness the percentage values for three type of exceptions lager than values in new approach, that because the number of mutants in group 1 is increased and appear early (small number of generation than GA without fitness).

The difference in percentage value for two approaches is: 3% in others exception, 3% in NDF exception and 13% in TMR exception.

Query 9

Table 24 Percentages of Mutants Exceptions Coverage in Query 9

	GA without fitness	GA with fitness
NDF	72%	76%
TMR	97%	97%
Others	100%	100%

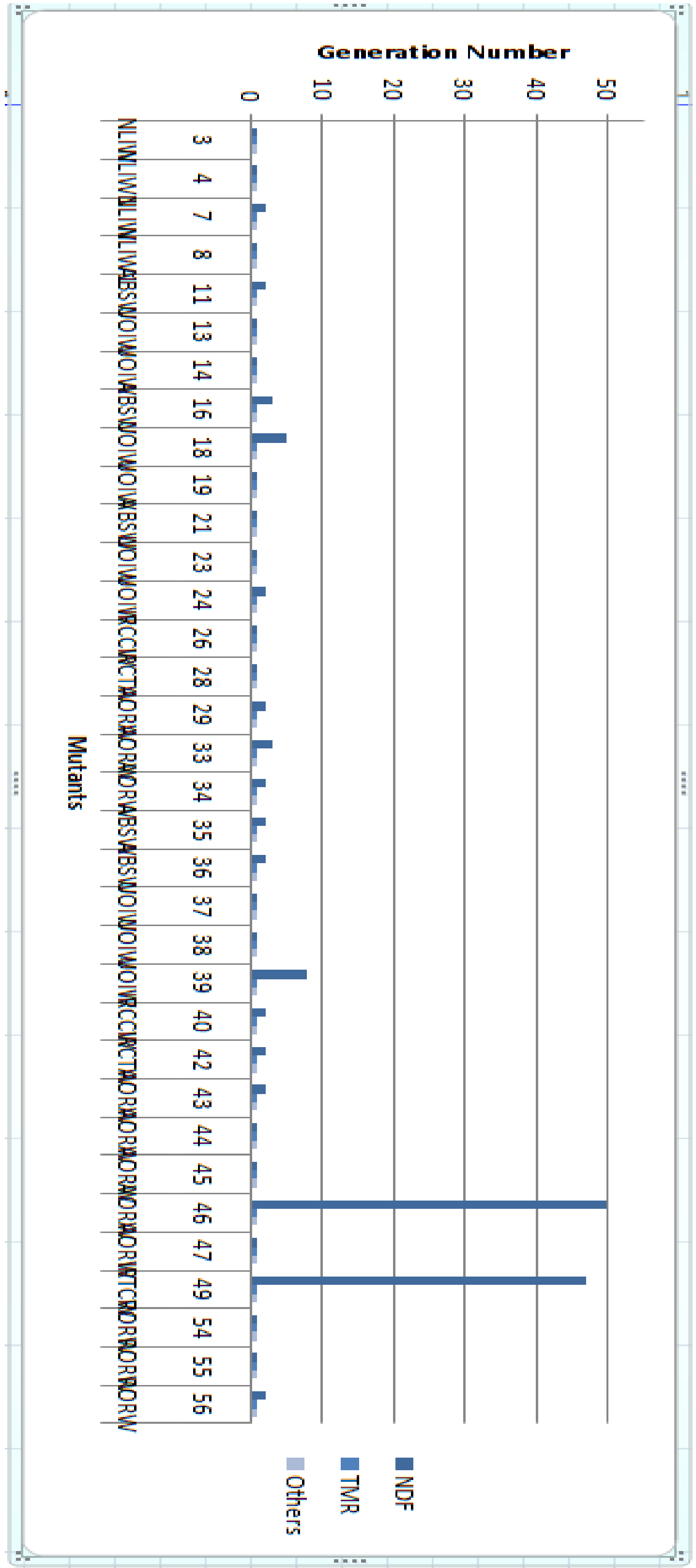

Figure 48 Group 1 Mutants Coverage in Query 9 for GA with fitness

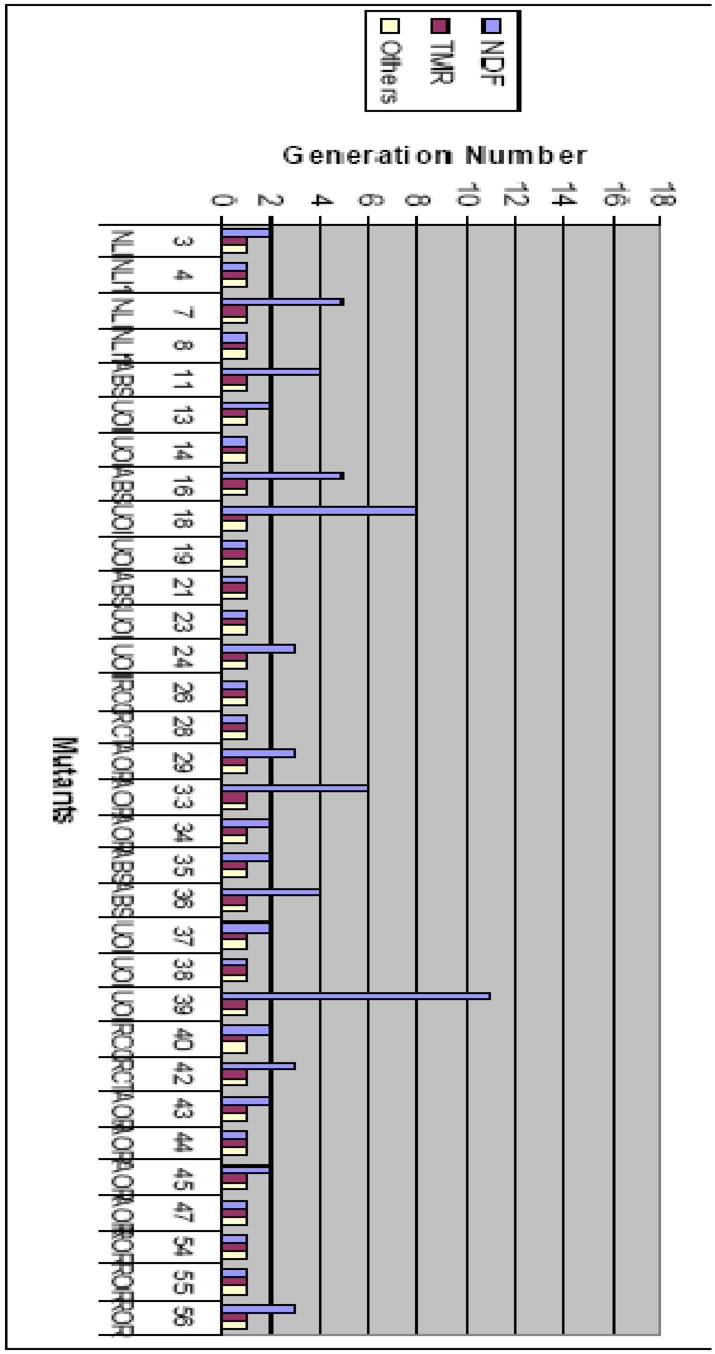

Figure 49 Group 1 Mutants Coverage in Query 9 for GA without Fitness

In query 9 there are many mutants covered in group 1 as appear in figures approve. Figures show that new approach in must mutants covered early than old one.

There are numbers of mutants appear in group1 in new approaches that make the coverage exception have best values for new approach.

Query 10

Table 25 Percentages of Mutants Exceptions Coverage in Query 10

	GA without fitness	GA with fitness
NDF	57%	57%
TMR	42%	44%
Others	100%	100%

Query 10 have same percentage values for NDF and others exception in two approach, that depend on our test schema and nature of query 10 that make the comparison of two approach not very clear. But there is 2% difference in ration for TMR exception.

Query 11

Table 26 Percentages of Mutants Exceptions Coverage in Query 11

	GA without fitness	GA with fitness
NDF	79%	79%
TMR	48%	59%
Others	94%	100%

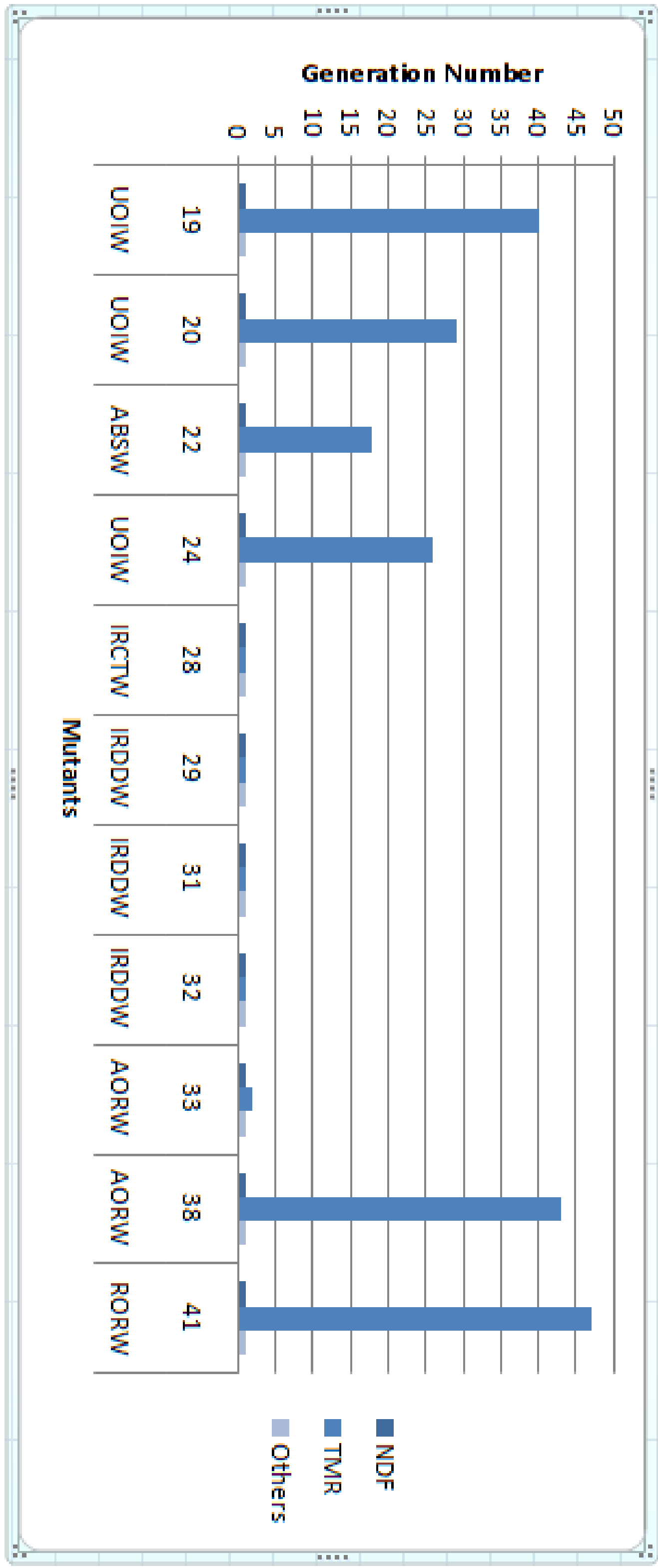

Figure 50 Group 1 Mutants Coverage in Query 11 for GA with fitness

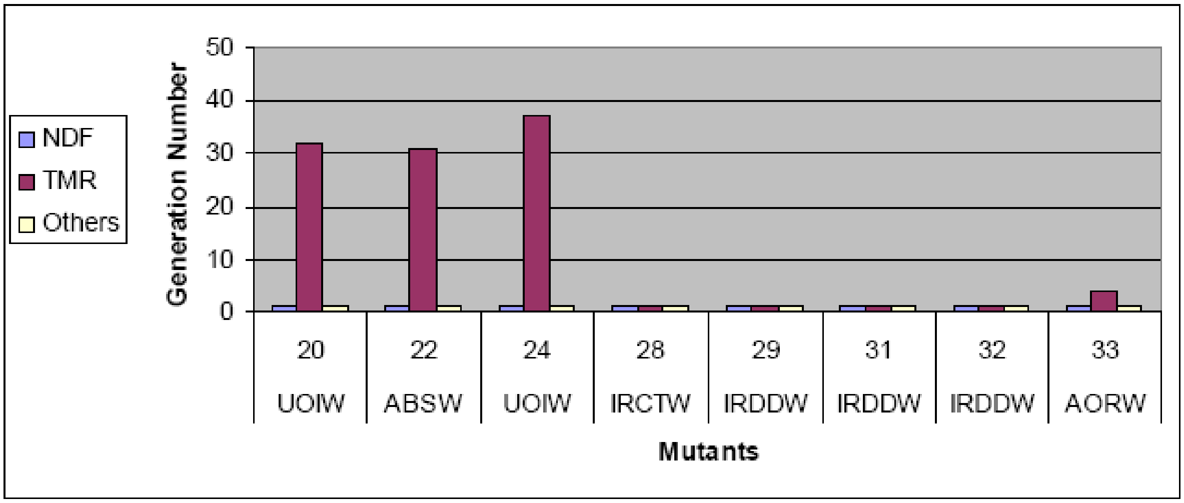

Figure 51 Group 1 Mutants Coverage in Query 11 for GA without Fitness

TMR in new approach have best value than old one with difference 11%. That because three mutants covered early using new approach and related to group 1. The new mutants appear in group 1 are: mutant 41, mutant 32 and mutant 19.

Query 12

Table 27 Percentages of Mutants Exceptions Coverage in Query 12

	GA without fitness	GA with fitness
NDF	82%	83%
TMR	35%	38%
Others	100%	100%

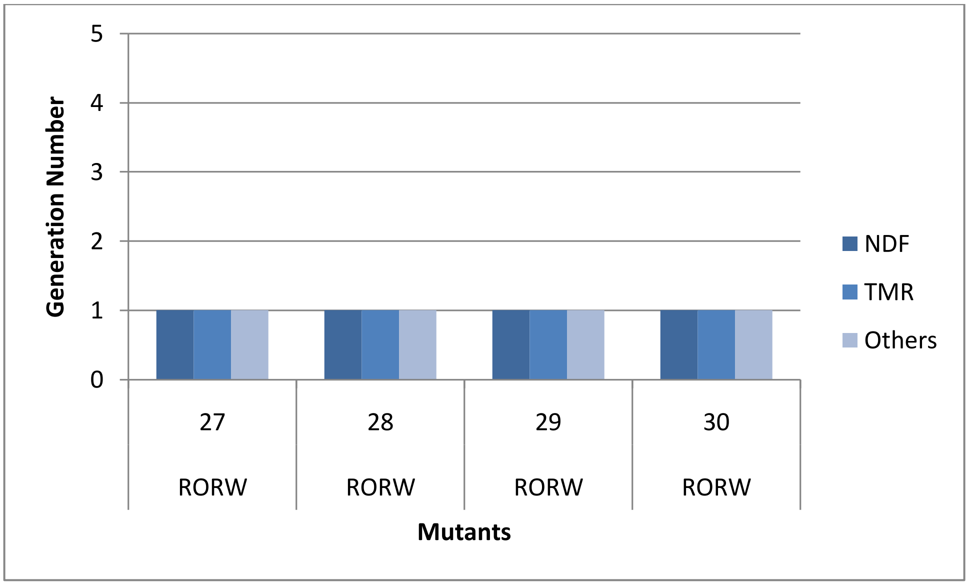

Figure 52 Group 1 Mutants Coverage in Query 12 for GA with fitness

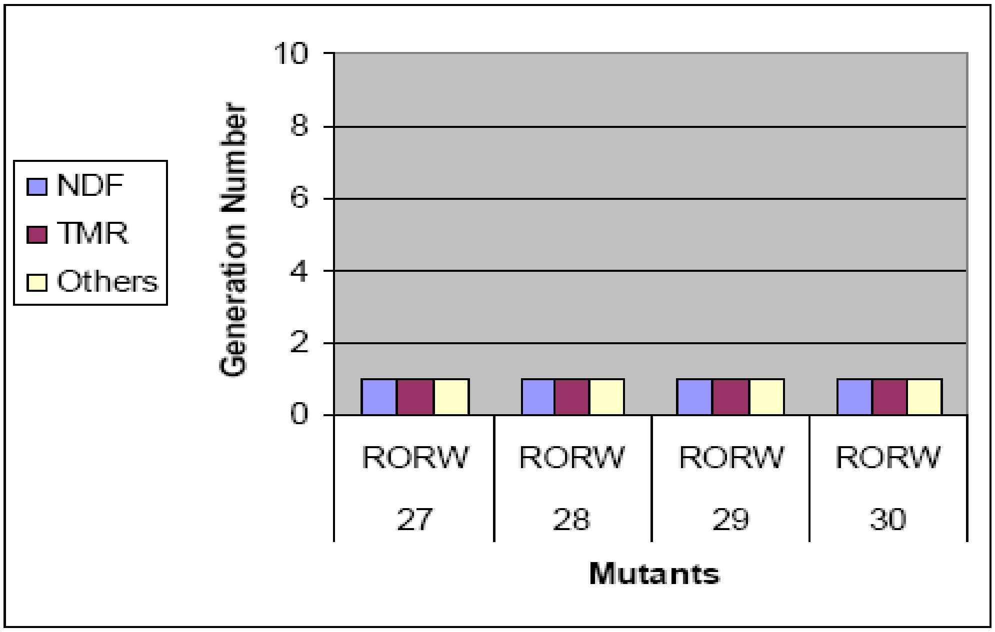

Figure 53 Group 1 Mutants Coverage in Query 12 for GA without Fitness

Table 43 show the difference between two approaches percentage values are very small (NDF 1% and TMR 3%). The data in HR schema with nature of query 12 main facture to make the difference of percentage very

small between approaches. All mutants appear early (from first generation) in two approaches as figures 52-53 show.

Query 13

Table 28 Percentages of Mutants Exceptions Coverage in Query 13

	GA without fitness	GA with fitness
NDF	93%	93%
TMR	23%	23%
Others	97%	100%

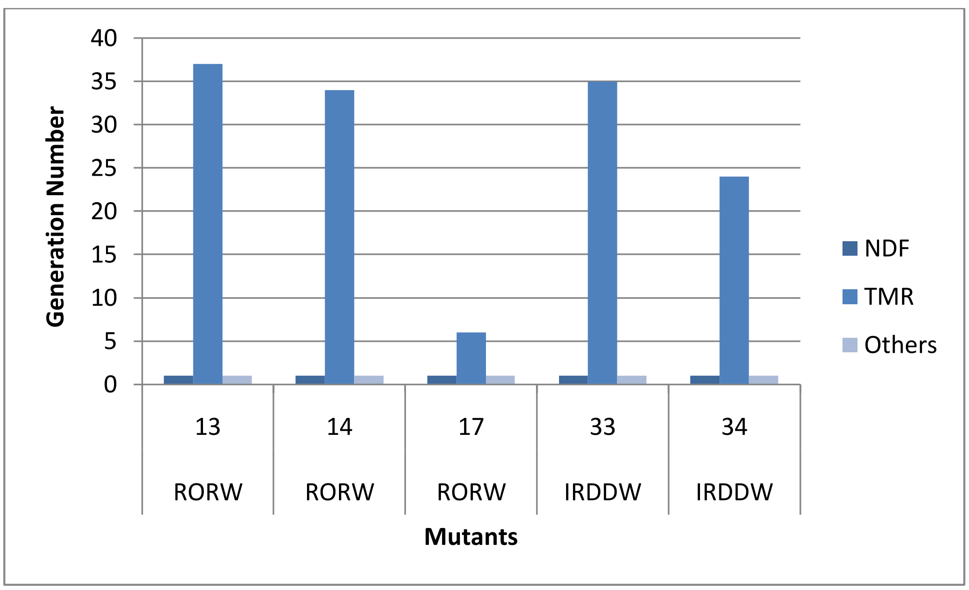

Figure 54 Group 1 Mutants Coverage in Query 13 for GA with fitness

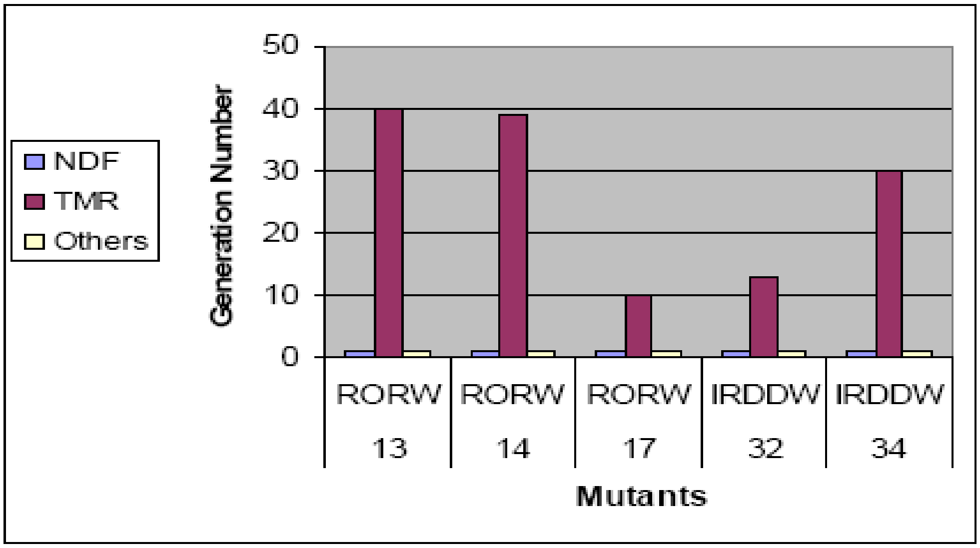

Figure 55 Group 1 Mutants Coverage in Query 13 for GA without Fitness

Same mutants appear in group 1 for two approaches but in difference with generation number. In our new approach every mutants appear early comparing with old one as figures above appear. The percentage of coverage in two approach for exceptions have same value except in other exception has difference value 3%.

Query 14

Table 29 Percentages of Mutants Exceptions Coverage in Query 14

	GA without fitness	GA with fitness
NDF	92%	92%
TMR	63%	64%
Others	81%	100%

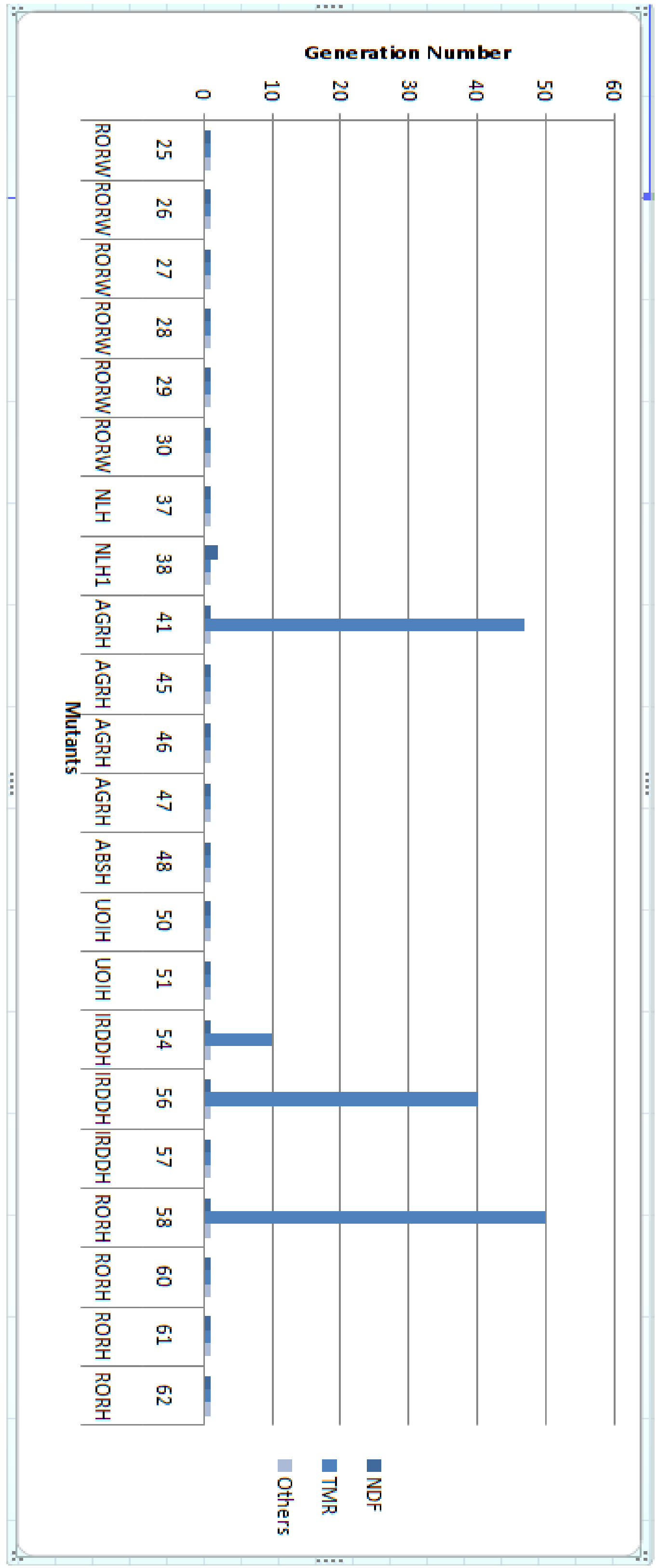

Figure 56 Group 1 Mutants Coverage in Query 14 for GA with fitness

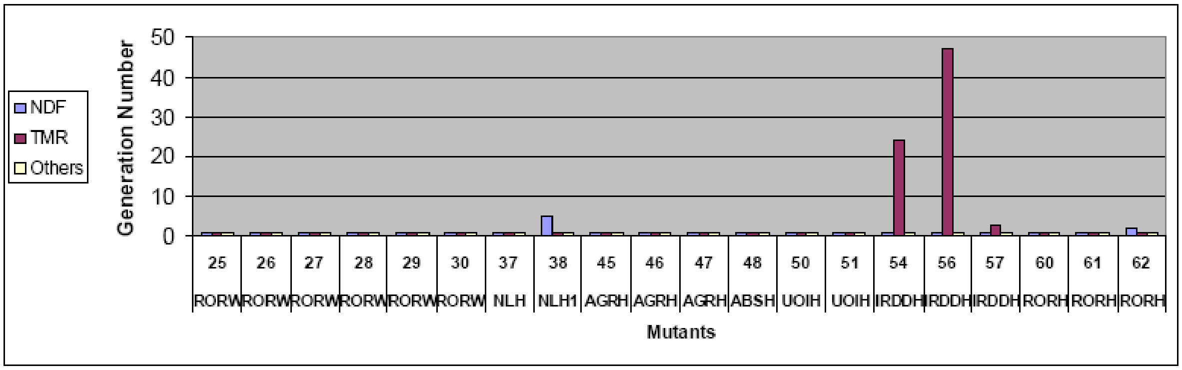

Figure 57 Group 1 Mutants Coverage in Query 14 for GA without Fitness

Two mutants in query 14 appear early (in group 1) in new approach make a difference in percentage values for two approaches. Difference equal 1% for TMR exception because M58 and M41 covered in less than 50 generation in new approaches. In other side other exception covered early in many mutants that make the difference 19% between two approaches.

Query 15

Table 30 Percentages of Mutants Exceptions Coverage in Query 15

	GA without fitness	GA with fitness
NDF	0%	0%
TMR	0%	0%
Others	100%	100%

As table above appears that NDF and TMR exceptions not covered any time, but others exception covered with percentage 100%. These values appear same for two approaches because the nature of query reflects these percentages and not change for any approach under this database schema. The percentage for NDF and TMR is 0%, this value maybe change under other database schema.

Query 16

Table 31 Percentages of Mutants Exceptions Coverage in Query 16

	GA without fitness	GA with fitness
NDF	85%	90%
TMR	35%	52%
Others	90%	100%

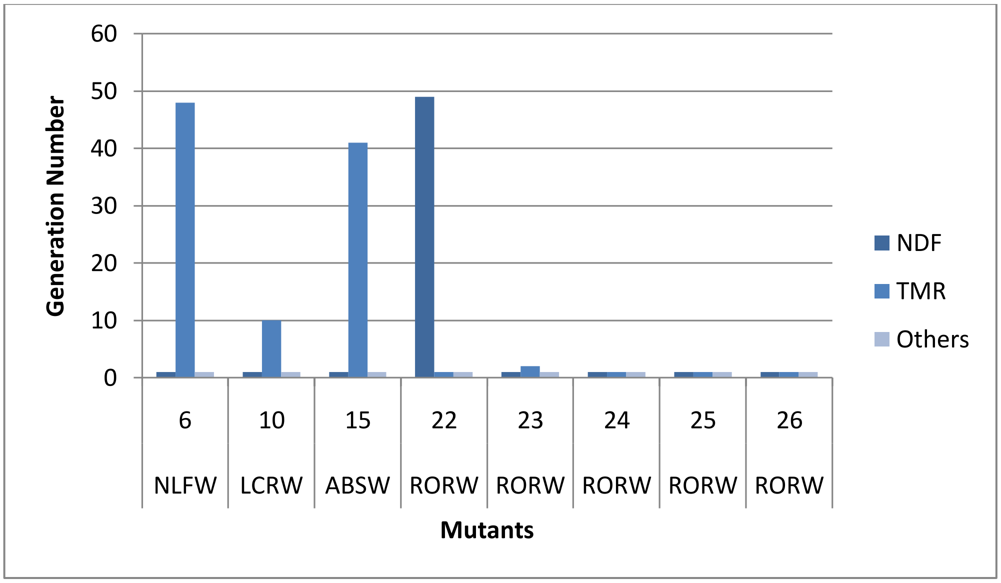

Figure 58 Group 1 Mutants Coverage in Query 16 for GA with fitness

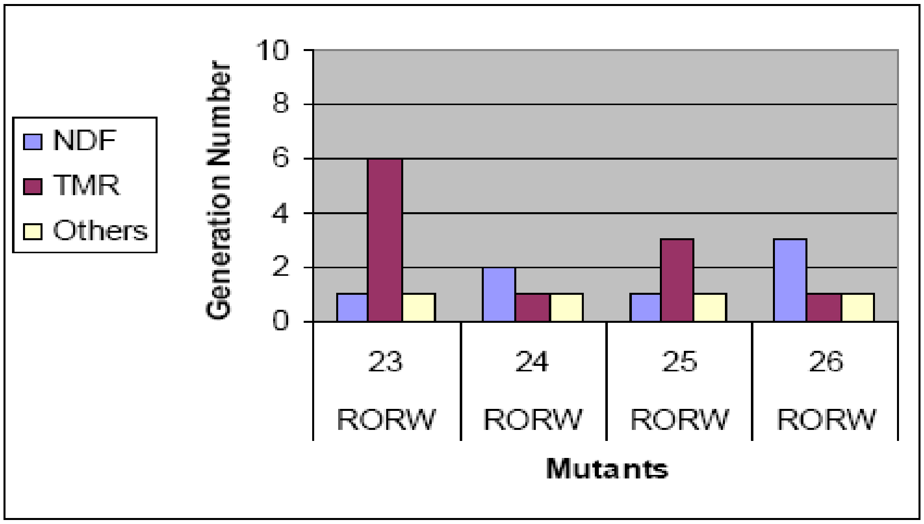

Figure 59 Group 1 Mutants Coverage in Query 16 for GA without Fitness

In GA without fitness there are four mutants appear in group 1.but in new approach there are other four mutants appear in group1 (M22, M15, M10 and M6). These four mutants affect the percentage of coverage exception in new approaches. The differences in exceptions are: 5% for NDF, 17% for TMR and 10% for other exception.

Query 17

Table 32 Percentages of Mutants Exceptions Coverage in Query 17

	GA without fitness	GA with fitness
NDF	75%	92%
TMR	24%	58%
Others	92%	100%

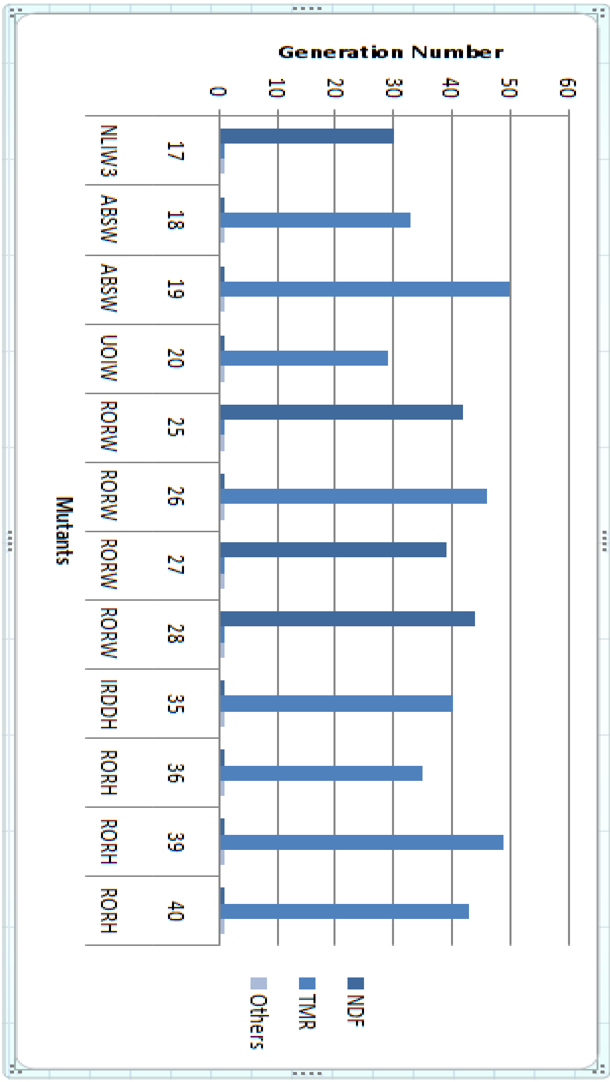

Figure 60 Group 1 Mutants Coverage in Query 17 for GA with fitness

Figure above appear that all mutants coved later(after large number of generation). That because query 17 with this schema and its data not easy to coved early (nature of query 17). In GA without fitness there is no mutants related to group 1 because all mutant related to group 3 that need generation more than 50 times to covered exceptions. But in new approach 12 mutants appear in group1 with very good values comparing with GA without fitness.

CHAPTER VI

Conclusion and Future Research

6.1 Conclusion of Proposal

In this thesis new system for software testing are created. That creates test cases automatically to cover three types of exceptions in oracle database. We used mutation testing and genetic algorithm that generate test case automatically. We used fitness of query that we execute in our program to test covering three types of exceptions that are No_Data_Found, Too_Many_Row and Others exceptions.

The fitness of SELECT statements is the main point in our proposed idea, and the range of data values related to columns in SELECT statement. Those minimum and maximum values for columns in query conditions in the query or its mutant used as a fitness value to guide GA to cover NDF exceptions. Also the aggregation function count of values of columns that related to query conditions used as guided of GA to cover TMR exceptions.

We used in our experiments HR schema in oracle database 10g. This schema is a sample database so we need to get more good result to test under schema that have large number of data to get more realistic result.

We also discuss the result of each query and which type of exceptions that covered. And explain the result for each mutant in any queries if it

covered the exceptions or have invisible paths. The higher percentage of coverage exception in our proposed idea that depends on fitness of queries for Others exceptions then NDF and TMR has lowest Percentage.

6.2 Future Research

In this research we create new technique to cover three types of exceptions, and to get best value there are some future works like: test this system that depends on the fitness of query in a large schema (e.g. Hospital Data Base Records), that makes the results more reality. In this system some query is filtered manually, like convert JOIN statement to WHERE seatmate to execute it under program, so in future this converts done automatically. Only three types of exceptions are tested here, while large database has more types of exceptions have to be tested in the future works. GA with fitness in our testing covered some datatype in database so in future hope to cover other datatype.

References

Chan M.Y, Cheung S.C. (1999), Testing database applications with SQL semantics, **In Proceedings of the 2nd International Symposium on Cooperative Database Systems for Advanced Applications,** Springer, Singapore pp. 363–374 .

Ram Chillarege (1999), software testing best practices, Center for software engineering IBM research.

Carsten Binnig, Donald Kossmann, Eric L. (2008), **Towards Automatic Test Database Generation,** IEEE Computer Society Technical Committee on Data Engineering.

Chays David and Deng Y. (2005). Testing Database Transactions with AGENDA**. In Proceedings of the 27th International Conference on Software Engineering (ICSE)**. IEEE Computer Society.

Chen Mingsong, Qiu Xiaokang, and Li Xuandong (2006), Automatic Test Case Generation for UML Activity Diagram.

Mrs.R. Jeevarathinam, Dr. Antony Selvadoss Thanamani (2010), TOWARDS TEST CASES GENERATION FROM SOFTWARE SPECIFICATIONS**,** R.Jeevarathinam et. al. / International Journal of Engineering Science and Technology.

Santosh K.S., Durga P.M., Rajib M. (2010). Test Case Generation Based on State and Activity Models, Journal of Object Technology.

Jiang S.J., Zhang Y.P ,Yan D.S ,Jiang Y.P. (2005). An Approach to Automatic Testing Exception Handling, **ACM SIGPLAN Notices**, Vol. 40(8).

W.K. Chan, S. C. Cheung, T.H. Tse. (2005). Fault-Based Testing of Database Application Programs with Conceptual Data Model. **in Proceedings of the 5th International Conference on Quality Software (QSIC 2005), IEEE Computer Society Press, Los Alamitos, California (2005).**

Yuetang Deng, Phyllis Frankl, David Chays (2005). Testing Database Transactions with AGENDA.

D. J. Berndt, A. Watkins.(2005) High Volume Software Testing using Genetic Algorithms**. Proceedings of the 38th Hawaii International Conference on System Sciences.**

Ezdehar A. N Jawabreh, Dr. Mohammad Alshraideh(2009).Using Mutation Testing to Generate Test Cases for Exceptions Handling Coverage in SQL Database.

Holland JH (1975). Adaptation in Natural and Artificial Systems; University of Michigan Press.

Haraty R.A., Mansour Nash'at, Daou Bassel (2002), Regression Testing of Database Applications, **Journal of Database Management**. 13(2) 2002 31-42.

Jiang S.J and Jiang Y.P. (2007).An Analysis Approach for Testing Exception Handling Programs**, ACM SIGPLAN Notices** ,Vol. 42 (4).

Khor S. and Grogono P. (2004). Using a Genetic Algorithm and Formal Concept Analysis to Generate Branch Coverage Test Data Automatically**, In the Proceedings of the 19th International Conference on Automated Software Engineering**.

Michael C. C., McGraw G. E. , Schatz, M. A.,Walton C. C.(1997). Genetic Algorithms for Dynamic Test Data Generation**, IEEE Computer Society**.

Tracey N., Clark J., Mander K. , McDermid J. (2000).Automated Test-Data Generation For Exception Conditions. **Software-Practice and Experience,** 30:61-79.

Tuya Javier, Suarez-Cabal M.J., de la Riva Claudio.(2006). Sqlmutation: A Tool To
Generate Mutants Of SQL Database Queries, **In Proceedings of Second Workshop on Mutation Analysis (Mutation 2006 - ISSRE Workshops 2006)**, IEEE Computer Society Press.

Wikipedia .The free encyclopedia ,[http://www. Wikipedia.org].Site last visited February 2011.

Chan, Cheung. (1999). Testing Database Applications with SQL Semantics.1999 363-374.

María José Suárez-Cabal, Javier Tuya.Using an SQL Coverage Measurement for Testing Database Applications.(2004). 2004 **ACM** 1-58113-855.

Carretero, Xhafa. (2005).Genetic Algorithm Based Schedulers for Grid Computing System. **ICIC International** 2005 ISSN 1349-4198.

Wu Xintao, Wang Yongge, Zheng Yuliang (2003)., Privacy preserving database
application testing. **Proceedings of the ACM Workshop on Privacy in Electronic Society**, ACM Press, New York, NY, USA, pp.118-128.

Zhang Jian, Xu C, Cheung S.C. (2001), Automatic generation of database instances for white-box testing.**Proceedings of the 25th Annual International Computer Software and Applications Conference**,IEEE Computer Society Press, Los Alamitos, CA, pp. 161–165.

Javier Tuya, Cabal, Claudio. (2009). Full predicate coverage for testing SQL database queries.

Appendix A
Sample of Queries Details

Query 1

SELECT * from employees where employee_id=?xi?

ID	Cat	Type	SubType	Mutated SQL
2	OR	ABS	ABSW	SELECT * FROM employees WHERE ABS(employee_id) = ?xi?
3	OR	ABS	ABSW	SELECT * FROM employees WHERE (-ABS(employee_id)) = ?xi?
4	OR	UOI	UOIW	SELECT * FROM employees WHERE ((employee_id)+1) = ?xi?
5	OR	UOI	UOIW	SELECT * FROM employees WHERE ((employee_id)-1) = ?xi?
6	OR	UOI	UOIW	SELECT * FROM employees WHERE (-(employee_id)) = ?xi?
7	IR	IRD	IRDDW	SELECT * FROM employees WHERE EMPLOYEES.SALARY = ?xi?
8	IR	IRD	IRDDW	SELECT * FROM employees WHERE EMPLOYEES.COMMISSION_PCT = ?xi?
9	IR	IRD	IRDDW	SELECT * FROM employees WHERE EMPLOYEES.MANAGER_ID = ?xi?
10	IR	IRD	IRDDW	SELECT * FROM employees WHERE EMPLOYEES.DEPARTMENT_ID = ?xi?
11	OR	ROR	RORW	SELECT * FROM employees WHERE employee_id <> ?xi?
12	OR	ROR	RORW	SELECT * FROM employees WHERE employee_id > ?xi?
13	OR	ROR	RORW	SELECT * FROM employees WHERE employee_id < ?xi?
14	OR	ROR	RORW	SELECT * FROM employees WHERE employee_id >= ?xi?

15	OR	ROR	RORW	SELECT * FROM employees WHERE employee_id <= ?xi?
16	OR	ROR	RORW	SELECT * FROM employees WHERE (1=1)
17	OR	ROR	RORW	SELECT * FROM employees WHERE (1=0)

Query 2

SELECT (employee_id) from employees where hire_date>?xu? order by department_id

ID	Cat	Type	SubType	Mutated SQL
11	OR	ROR	RORW	SELECT (employee_id) FROM employees WHERE hire_date = ?xu? ORDER BY department_id
12	OR	ROR	RORW	SELECT (employee_id) FROM employees WHERE hire_date <> ?xu? ORDER BY department_id
13	OR	ROR	RORW	SELECT (employee_id) FROM employees WHERE hire_date < ?xu? ORDER BY department_id
14	OR	ROR	RORW	SELECT (employee_id) FROM employees WHERE hire_date >= ?xu? ORDER BY department_id
15	OR	ROR	RORW	SELECT (employee_id) FROM employees WHERE hire_date <= ?xu? ORDER BY department_id

Query 3

SELECT manager_id from departments where location_id=?xi? or department_id=?di?

ID	Cat	Type	SubType	Mutated SQL
12	NL	NLI	NLIW	SELECT manager_id FROM departments WHERE (DEPARTMENTS.LOCATION_ID IS NULL OR location_id = ?xi?) OR department_id = ?di?
13	NL	NLO	NLIW1	SELECT manager_id FROM departments WHERE

				(DEPARTMENTS.LOCATION_ID IS NULL OR NOT location_id = ?xi?) OR department_id = ?di?
14	NL	NLO	NLIW2	SELECT manager_id FROM departments WHERE (DEPARTMENTS.LOCATION_ID IS NULL) OR department_id = ?di?
15	NL	NLO	NLIW3	SELECT manager_id FROM departments WHERE (DEPARTMENTS.LOCATION_ID IS NOT NULL) OR department_id = ?di?
16	OR	ABS	ABSW	SELECT manager_id FROM departments WHERE ABS(location_id) = ?xi? OR department_id = ?di?
17	OR	ABS	ABSW	SELECT manager_id FROM departments WHERE (-ABS(location_id)) = ?xi? OR department_id = ?di?
18	OR	UOI	UOIW	SELECT manager_id FROM departments WHERE ((location_id)+1) = ?xi? OR department_id = ?di?
19	OR	UOI	UOIW	SELECT manager_id FROM departments WHERE ((location_id)-1) = ?xi? OR department_id = ?di?
20	OR	UOI	UOIW	SELECT manager_id FROM departments WHERE (-(location_id)) = ?xi? OR department_id = ?di?
21	IR	IRC	IRCCW	SELECT manager_id FROM departments WHERE DEPARTMENTS.MANAGER_ID = ?xi? OR department_id = ?di?
22	IR	IRC	IRCCW	SELECT manager_id FROM departments WHERE DEPARTMENTS.DEPARTMENT_ID = ?xi? OR department_id = ?di?
23	OR	ROR	RORW	SELECT manager_id FROM departments WHERE location_id <> ?xi? OR department_id = ?di?
24	OR	ROR	RORW	SELECT manager_id FROM departments WHERE location_id > ?xi? OR department_id = ?di?
25	OR	ROR	RORW	SELECT manager_id FROM departments WHERE location_id < ?xi? OR department_id = ?di?
26	OR	ROR	RORW	SELECT manager_id FROM departments WHERE location_id >= ?xi? OR department_id = ?di?
27	OR	ROR	RORW	SELECT manager_id FROM departments WHERE location_id <= ?xi? OR department_id = ?di?
28	OR	ROR	RORW	SELECT manager_id FROM departments WHERE (1=1) OR department_id = ?di?
29	OR	ROR	RORW	SELECT manager_id FROM departments WHERE

				(1=0) OR department_id = ?di?
30	IR	IRP	IRPCW	SELECT manager_id FROM departments WHERE location_id = DEPARTMENTS.MANAGER_ID OR department_id = ?di?
31	IR	IRP	IRPCW	SELECT manager_id FROM departments WHERE location_id = DEPARTMENTS.DEPARTMENT_ID OR department_id = ?di?
32	IR	IRP	IRPPW	SELECT manager_id FROM departments WHERE location_id = ?di? OR department_id = ?di?
33	OR	LCR	LCRW	SELECT manager_id FROM departments WHERE location_id = ?xi? AND department_id = ?di?
34	OR	LCR	LCRW	SELECT manager_id FROM departments WHERE (1=1)
35	OR	LCR	LCRW	SELECT manager_id FROM departments WHERE (1=0)
36	OR	LCR	LCRW	SELECT manager_id FROM departments WHERE location_id = ?xi?
37	OR	LCR	LCRW	SELECT manager_id FROM departments WHERE department_id = ?di?

Query 4

SELECT e.employee_id,j.max_salary from employees e, jobs j where

e.job_id=j.job_id and e.commission_pct<=?xd?

ID	Cat	Type	SubType	Mutated SQL
22	IR	IRD	IRDDS	SELECT e.employee_id , j.MIN_SALARY FROM employees e , jobs j WHERE e.job_id = j.job_id AND e.commission_pct <= ?xd?
23	SC	JOI	JOIN	SELECT e.employee_id , j.max_salary FROM employees e LEFT JOIN jobs j ON e.job_id = j.job_id WHERE e.commission_pct <= ?xd?
27	NL	NLO	NLIW1	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.job_id = j.job_id AND

				(e.COMMISSION_PCT IS NULL OR NOT e.commission_pct <= ?xd?)
28	NL	NLO	NLIW2	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.job_id = j.job_id AND (e.COMMISSION_PCT IS NULL)
29	NL	NLO	NLIW3	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.job_id = j.job_id AND (e.COMMISSION_PCT IS NOT NULL)
30	IR	IRD	IRDDW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.FIRST_NAME = j.job_id AND e.commission_pct <= ?xd?
31	IR	IRD	IRDDW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.LAST_NAME = j.job_id AND e.commission_pct <= ?xd?
32	IR	IRD	IRDDW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.EMAIL = j.job_id AND e.commission_pct <= ?xd?
33	IR	IRD	IRDDW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.PHONE_NUMBER = j.job_id AND e.commission_pct <= ?xd?
34	OR	ROR	RORW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.job_id <> j.job_id AND e.commission_pct <= ?xd?
35	OR	ROR	RORW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.job_id > j.job_id AND e.commission_pct <= ?xd?
36	OR	ROR	RORW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.job_id < j.job_id AND e.commission_pct <= ?xd?
37	OR	ROR	RORW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.job_id >= j.job_id AND e.commission_pct <= ?xd?
38	OR	ROR	RORW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.job_id <= j.job_id AND e.commission_pct <= ?xd?
39	OR	ROR	RORW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE (1=1) AND

				e.commission_pct <= ?xd?
43	OR	LCR	LCRW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE (1=1)
44	OR	LCR	LCRW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE (1=0)
45	OR	LCR	LCRW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.job_id = j.job_id
46	OR	LCR	LCRW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.commission_pct <= ?xd?
47	OR	ABS	ABSW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.job_id = j.job_id AND ABS(e.commission_pct) <= ?xd?
48	OR	ABS	ABSW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.job_id = j.job_id AND (-ABS(e.commission_pct)) <= ?xd?
49	OR	UOI	UOIW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.job_id = j.job_id AND ((e.commission_pct)+1) <= ?xd?
50	OR	UOI	UOIW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.job_id = j.job_id AND ((e.commission_pct)-1) <= ?xd?
51	OR	UOI	UOIW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.job_id = j.job_id AND (-(e.commission_pct)) <= ?xd?
52	IR	IRC	IRCCW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.job_id = j.job_id AND e.EMPLOYEE_ID <= ?xd?
53	IR	IRC	IRCCW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.job_id = j.job_id AND j.MAX_SALARY <= ?xd?
54	IR	IRD	IRDDW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.job_id = j.job_id AND e.SALARY <= ?xd?
55	IR	IRD	IRDDW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.job_id = j.job_id AND e.MANAGER_ID <= ?xd?

56	IR	IRD	IRDDW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.job_id = j.job_id AND e.DEPARTMENT_ID <= ?xd?
57	OR	ROR	RORW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.job_id = j.job_id AND e.commission_pct = ?xd?
58	OR	ROR	RORW	SELECT e.employee_id , j.max_salary FROM employees e , jobs j WHERE e.job_id = j.job_id AND e.commission_pct <> ?xd?

Query 5

SELECT e.first_name from employees e left join job_history h on

e.employee_id=h.employee_id and h.start_date=?xu?

ID	Cat	Type	SubType	Mutated SQL
7	SC	JOI	JOIN	SELECT e.first_name FROM employees e RIGHT JOIN job_history h ON e.employee_id = h.employee_id AND h.start_date = ?xu?
8	SC	JOI	JOIN	SELECT e.first_name FROM employees e INNER JOIN job_history h ON e.employee_id = h.employee_id AND h.start_date = ?xu?
9	SC	JOI	JOIN	SELECT e.first_name FROM employees e FULL OUTER JOIN job_history h ON e.employee_id = h.employee_id AND h.start_date = ?xu?
11	OR	ABS	ABSJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON ABS(e.employee_id) = h.employee_id AND h.start_date = ?xu?
12	OR	ABS	ABSJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON (-ABS(e.employee_id)) = h.employee_id AND h.start_date = ?xu?
13	OR	UOI	UOIJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON ((e.employee_id)+1) = h.employee_id AND h.start_date = ?xu?
14	OR	UOI	UOIJ	SELECT e.first_name FROM employees e LEFT

				JOIN job_history h ON ((e.employee_id)-1) = h.employee_id AND h.start_date = ?xu?
15	OR	UOI	UOIJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON (-(e.employee_id)) = h.employee_id AND h.start_date = ?xu?
16	IR	IRD	IRDDJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON e.SALARY = h.employee_id AND h.start_date = ?xu?
17	IR	IRD	IRDDJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON e.COMMISSION_PCT = h.employee_id AND h.start_date = ?xu?
18	IR	IRD	IRDDJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON e.MANAGER_ID = h.employee_id AND h.start_date = ?xu?
19	IR	IRD	IRDDJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON e.DEPARTMENT_ID = h.employee_id AND h.start_date = ?xu?
20	OR	ROR	RORJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON e.employee_id <> h.employee_id AND h.start_date = ?xu?
21	OR	ROR	RORJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON e.employee_id > h.employee_id AND h.start_date = ?xu?
22	OR	ROR	RORJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON e.employee_id < h.employee_id AND h.start_date = ?xu?
23	OR	ROR	RORJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON e.employee_id >= h.employee_id AND h.start_date = ?xu?
24	OR	ROR	RORJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON e.employee_id <= h.employee_id AND h.start_date = ?xu?
25	OR	ROR	RORJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON (1=1) AND h.start_date = ?xu?
26	OR	ROR	RORJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON (1=0) AND h.start_date = ?xu?
27	OR	ABS	ABSJ	SELECT e.first_name FROM employees e LEFT

				JOIN job_history h ON e.employee_id = ABS(h.employee_id) AND h.start_date = ?xu?
28	OR	ABS	ABSJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON e.employee_id = (-ABS(h.employee_id)) AND h.start_date = ?xu?
29	OR	UOI	UOIJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON e.employee_id = ((h.employee_id)+1) AND h.start_date = ?xu?
30	OR	UOI	UOIJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON e.employee_id = ((h.employee_id)-1) AND h.start_date = ?xu?
31	OR	UOI	UOIJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON e.employee_id = (-(h.employee_id)) AND h.start_date = ?xu?
32	IR	IRD	IRDDJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON e.employee_id = h.DEPARTMENT_ID AND h.start_date = ?xu?
33	OR	LCR	LCRJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON e.employee_id = h.employee_id OR h.start_date = ?xu?
37	OR	LCR	LCRJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON h.start_date = ?xu?
38	IR	IRD	IRDDJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON e.employee_id = h.employee_id AND h.END_DATE = ?xu?
39	OR	ROR	RORJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON e.employee_id = h.employee_id AND h.start_date <> ?xu?
40	OR	ROR	RORJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON e.employee_id = h.employee_id AND h.start_date > ?xu?
41	OR	ROR	RORJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON e.employee_id = h.employee_id AND h.start_date < ?xu?
42	OR	ROR	RORJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON e.employee_id = h.employee_id AND h.start_date >= ?xu?

43	OR	ROR	RORJ	SELECT e.first_name FROM employees e LEFT JOIN job_history h ON e.employee_id = h.employee_id AND h.start_date <= ?xu?

Query 6

SELECT job_id,avg(salary) from employees group by job_id having sum(salary)<?xi?

ID	Cat	Type	SubType	Mutated SQL
26	NL	NLI	NLIH	SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING (SUM(EMPLOYEES.SALARY) IS NULL OR SUM(salary) < ?xi?)
27				SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING (SUM(EMPLOYEES.SALARY) IS NULL OR NOT SUM(salary) < ?xi?)
30	SC	AGR	AGRH	SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING COUNT(salary) < ?xi?
31	SC	AGR	AGRH	SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING COUNT(DISTINCT salary) < ?xi?
32	SC	AGR	AGRH	SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING SUM(DISTINCT salary) < ?xi?
33	SC	AGR	AGRH	SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING MAX(salary) < ?xi?
34	SC	AGR	AGRH	SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING MIN(salary) < ?xi?
35	SC	AGR	AGRH	SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING AVG(salary) < ?xi?
36	SC	AGR	AGRH	SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING AVG(DISTINCT salary) < ?xi?

37	OR	ABS	ABSH	SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING SUM(ABS(salary)) < ?xi?
38	OR	ABS	ABSH	SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING SUM((-ABS(salary))) < ?xi?
39	OR	UOI	UOIH	SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING SUM(((salary)+1)) < ?xi?
40	OR	UOI	UOIH	SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING SUM(((salary)-1)) < ?xi?
41	OR	UOI	UOIH	SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING SUM((-(salary))) < ?xi?
42	IR	IRC	IRCPH	SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING SUM(?xi?) < ?xi?
43	IR	IRD	IRDDH	SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING SUM(EMPLOYEES.EMPLOYEE_ID) < ?xi?
44	IR	IRD	IRDDH	SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING SUM(EMPLOYEES.COMMISSION_PCT) < ?xi?
45	IR	IRD	IRDDH	SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING SUM(EMPLOYEES.MANAGER_ID) < ?xi?
46	IR	IRD	IRDDH	SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING SUM(EMPLOYEES.DEPARTMENT_ID) < ?xi?
47	OR	ROR	RORH	SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING SUM(salary) = ?xi?
48	OR	ROR	RORH	SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING SUM(salary) <> ?xi?
49	OR	ROR	RORH	SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING SUM(salary) > ?xi?
50	OR	ROR	RORH	SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING SUM(salary) >= ?xi?

51	OR	ROR	RORH	SELECT job_id , AVG(salary) FROM employees GROUP BY job_id HAVING SUM(salary) <= ?xi?

Query 7

SELECT job_id from job_history where end_date between ?xu? and ?yu?

ID	Cat	Type	SubType	Mutated SQL
4	IR	IRD	IRDDW	SELECT job_id FROM job_history WHERE JOB_HISTORY.START_DATE BETWEEN ?xu? AND ?yu?
5	OR	BTW	BTWW	SELECT job_id FROM job_history WHERE end_date NOT BETWEEN ?xu? AND ?yu?
7	OR	BTW	BTWW	SELECT job_id FROM job_history WHERE ((end_date > ?xu?) AND (end_date <= ?yu?))
8	OR	BTW	BTWW	SELECT job_id FROM job_history WHERE ((end_date >= ?xu?) AND (end_date < ?yu?))
9	IR	IRP	IRPCW	SELECT job_id FROM job_history WHERE end_date BETWEEN JOB_HISTORY.END_DATE AND ?yu?
11	IR	IRP	IRPCW	SELECT job_id FROM job_history WHERE end_date BETWEEN ?xu? AND JOB_HISTORY.END_DATE

Query 8

SELECT region_name from countries c , regions r where

r.region_id=c.region_id and c.country_id=?xc?

ID	Cat	Type	SubType	Mutated SQL
5	SC	JOI	JOIN	SELECT region_name FROM countries c LEFT JOIN regions r ON r.region_id = c.region_id WHERE c.country_id = ?xc?
7	SC	JOI	JOIN	SELECT region_name FROM countries c FULL OUTER JOIN regions r ON r.region_id = c.region_id

				WHERE c.country_id = ?xc?
8	NL	NLI	NLIW	SELECT region_name FROM countries c , regions r WHERE (c.REGION_ID IS NULL OR r.region_id = c.region_id) AND c.country_id = ?xc?
9	NL	NLO	NLIW1	SELECT region_name FROM countries c , regions r WHERE (c.REGION_ID IS NULL OR NOT r.region_id = c.region_id) AND c.country_id = ?xc?
10	NL	NLO	NLIW2	SELECT region_name FROM countries c , regions r WHERE (c.REGION_ID IS NULL) AND c.country_id = ?xc?
11	NL	NLO	NLIW3	SELECT region_name FROM countries c , regions r WHERE (c.REGION_ID IS NOT NULL) AND c.country_id = ?xc?
12	OR	ABS	ABSW	SELECT region_name FROM countries c , regions r WHERE ABS(r.region_id) = c.region_id AND c.country_id = ?xc?
13	OR	ABS	ABSW	SELECT region_name FROM countries c , regions r WHERE (-ABS(r.region_id)) = c.region_id AND c.country_id = ?xc?
14	OR	UOI	UOIW	SELECT region_name FROM countries c , regions r WHERE ((r.region_id)+1) = c.region_id AND c.country_id = ?xc?
15	OR	UOI	UOIW	SELECT region_name FROM countries c , regions r WHERE ((r.region_id)-1) = c.region_id AND c.country_id = ?xc?
16	OR	UOI	UOIW	SELECT region_name FROM countries c , regions r WHERE (-(r.region_id)) = c.region_id AND c.country_id = ?xc?
17	OR	ROR	RORW	SELECT region_name FROM countries c , regions r WHERE r.region_id <> c.region_id AND c.country_id = ?xc?
18	OR	ROR	RORW	SELECT region_name FROM countries c , regions r WHERE r.region_id > c.region_id AND c.country_id = ?xc?
19	OR	ROR	RORW	SELECT region_name FROM countries c , regions r WHERE r.region_id < c.region_id AND c.country_id = ?xc?

20	OR	ROR	RORW	SELECT region_name FROM countries c , regions r WHERE r.region_id >= c.region_id AND c.country_id = ?xc?
21	OR	ROR	RORW	SELECT region_name FROM countries c , regions r WHERE r.region_id <= c.region_id AND c.country_id = ?xc?
22	OR	ROR	RORW	SELECT region_name FROM countries c , regions r WHERE (1=1) AND c.country_id = ?xc?
23	OR	ROR	RORW	SELECT region_name FROM countries c , regions r WHERE (1=0) AND c.country_id = ?xc?
24	OR	ABS	ABSW	SELECT region_name FROM countries c , regions r WHERE r.region_id = ABS(c.region_id) AND c.country_id = ?xc?
25	OR	ABS	ABSW	SELECT region_name FROM countries c , regions r WHERE r.region_id = (-ABS(c.region_id)) AND c.country_id = ?xc?
26	OR	UOI	UOIW	SELECT region_name FROM countries c , regions r WHERE r.region_id = ((c.region_id)+1) AND c.country_id = ?xc?
30	OR	LCR	LCRW	SELECT region_name FROM countries c , regions r WHERE (1=1)
31	OR	LCR	LCRW	SELECT region_name FROM countries c , regions r WHERE (1=0)
32	OR	LCR	LCRW	SELECT region_name FROM countries c , regions r WHERE r.region_id = c.region_id
33	OR	LCR	LCRW	SELECT region_name FROM countries c , regions r WHERE c.country_id = ?xc?
34	IR	IRC	IRCCW	SELECT region_name FROM countries c , regions r WHERE r.region_id = c.region_id AND r.REGION_NAME = ?xc?
35	IR	IRD	IRDDW	SELECT region_name FROM countries c , regions r WHERE r.region_id = c.region_id AND c.COUNTRY_NAME = ?xc?
36	OR	ROR	RORW	SELECT region_name FROM countries c , regions r WHERE r.region_id = c.region_id AND c.country_id <> ?xc?
37	OR	ROR	RORW	SELECT region_name FROM countries c , regions r

				WHERE r.region_id = c.region_id AND c.country_id > ?xc?

Query 9

SELECT job_title from jobs where (max_salary + min_salary)/2 <?xi?

ID	Cat	Type	SubType	Mutated SQL
3	NL	NLI	NLIW	SELECT job_title FROM jobs WHERE (JOBS.MAX_SALARY IS NULL OR (max_salary + min_salary) / 2 < ?xi?)
4	NL	NLO	NLIW1	SELECT job_title FROM jobs WHERE (JOBS.MAX_SALARY IS NULL OR NOT (max_salary + min_salary) / 2 < ?xi?)
5	NL	NLO	NLIW2	SELECT job_title FROM jobs WHERE (JOBS.MAX_SALARY IS NULL)
6	NL	NLO	NLIW3	SELECT job_title FROM jobs WHERE (JOBS.MAX_SALARY IS NOT NULL)
7	NL	NLI	NLIW	SELECT job_title FROM jobs WHERE (JOBS.MIN_SALARY IS NULL OR (max_salary + min_salary) / 2 < ?xi?)
8	NL	NLO	NLIW1	SELECT job_title FROM jobs WHERE (JOBS.MIN_SALARY IS NULL OR NOT (max_salary + min_salary) / 2 < ?xi?)
11	OR	ABS	ABSW	SELECT job_title FROM jobs WHERE ABS((max_salary + min_salary) / 2) < ?xi?
12	OR	ABS	ABSW	SELECT job_title FROM jobs WHERE (- ABS((max_salary + min_salary) / 2)) < ?xi?
13	OR	UOI	UOIW	SELECT job_title FROM jobs WHERE (((max_salary + min_salary) / 2)+1) < ?xi?
14	OR	UOI	UOIW	SELECT job_title FROM jobs WHERE (((max_salary + min_salary) / 2)-1) < ?xi?
15	OR	UOI	UOIW	SELECT job_title FROM jobs WHERE (-((max_salary + min_salary) / 2)) < ?xi?
16	OR	ABS	ABSW	SELECT job_title FROM jobs WHERE (ABS(

				max_salary + min_salary)) / 2 < ?xi?
17	OR	ABS	ABSW	SELECT job_title FROM jobs WHERE ((- ABS(max_salary + min_salary))) / 2 < ?xi?
18	OR	UOI	UOIW	SELECT job_title FROM jobs WHERE (((max_salary + min_salary)+1)) / 2 < ?xi?
19	OR	UOI	UOIW	SELECT job_title FROM jobs WHERE (((max_salary + min_salary)-1)) / 2 < ?xi?
20	OR	UOI	UOIW	SELECT job_title FROM jobs WHERE ((-(max_salary + min_salary))) / 2 < ?xi?
21	OR	ABS	ABSW	SELECT job_title FROM jobs WHERE (ABS(max_salary) + min_salary) / 2 < ?xi?
22	OR	ABS	ABSW	SELECT job_title FROM jobs WHERE ((-ABS(max_salary)) + min_salary) / 2 < ?xi?
23	OR	UOI	UOIW	SELECT job_title FROM jobs WHERE (((max_salary)+1) + min_salary) / 2 < ?xi?
24	OR	UOI	UOIW	SELECT job_title FROM jobs WHERE (((max_salary)-1) + min_salary) / 2 < ?xi?
25	OR	UOI	UOIW	SELECT job_title FROM jobs WHERE ((-(max_salary)) + min_salary) / 2 < ?xi?
26	IR	IRC	IRCCW	SELECT job_title FROM jobs WHERE (JOBS.MIN_SALARY + min_salary) / 2 < ?xi?
28	IR	IRC	IRCTW	SELECT job_title FROM jobs WHERE (2 + min_salary) / 2 < ?xi?
29	OR	AOR	AORW	SELECT job_title FROM jobs WHERE (max_salary - min_salary) / 2 < ?xi?
30	OR	AOR	AORW	SELECT job_title FROM jobs WHERE (max_salary * min_salary) / 2 < ?xi?
31	OR	AOR	AORW	SELECT job_title FROM jobs WHERE (max_salary / min_salary) / 2 < ?xi?
32	OR	AOR	AORW	SELECT job_title FROM jobs WHERE (max_salary) / 2 < ?xi?
33	OR	AOR	AORW	SELECT job_title FROM jobs WHERE (min_salary) / 2 < ?xi?
34	OR	ABS	ABSW	SELECT job_title FROM jobs WHERE (max_salary + ABS(min_salary)) / 2 < ?xi?
35	OR	ABS	ABSW	SELECT job_title FROM jobs WHERE (max_salary + (-ABS(min_salary))) / 2 < ?xi?

36	OR	UOI	UOIW	SELECT job_title FROM jobs WHERE (max_salary + ((min_salary)+1)) / 2 < ?xi?
37	OR	UOI	UOIW	SELECT job_title FROM jobs WHERE (max_salary + ((min_salary)-1)) / 2 < ?xi?
38	OR	UOI	UOIW	SELECT job_title FROM jobs WHERE (max_salary + (-(min_salary))) / 2 < ?xi?
39	IR	IRC	IRCCW	SELECT job_title FROM jobs WHERE (max_salary + JOBS.MAX_SALARY) / 2 < ?xi?
40	IR	IRC	IRCPW	SELECT job_title FROM jobs WHERE (max_salary + ?xi?) / 2 < ?xi?
42	OR	AOR	AORW	SELECT job_title FROM jobs WHERE (max_salary + min_salary) + 2 < ?xi?
43	OR	AOR	AORW	SELECT job_title FROM jobs WHERE (max_salary + min_salary) - 2 < ?xi?
44	OR	AOR	AORW	SELECT job_title FROM jobs WHERE (max_salary + min_salary) * 2 < ?xi?
45	OR	AOR	AORW	SELECT job_title FROM jobs WHERE (max_salary + min_salary) < ?xi?
46	OR	AOR	AORW	SELECT job_title FROM jobs WHERE 2 < ?xi?
47	IR	IRT	IRTCW	SELECT job_title FROM jobs WHERE (max_salary + min_salary) / JOBS.MAX_SALARY < ?xi?
48	IR	IRT	IRTCW	SELECT job_title FROM jobs WHERE (max_salary + min_salary) / JOBS.MIN_SALARY < ?xi?
49	IR	IRT	IRTPW	SELECT job_title FROM jobs WHERE (max_salary + min_salary) / ?xi? < ?xi?
50	OR	ROR	RORW	SELECT job_title FROM jobs WHERE (max_salary + min_salary) / 2 = ?xi?
52	OR	ROR	RORW	SELECT job_title FROM jobs WHERE (max_salary + min_salary) / 2 > ?xi?
53	OR	ROR	RORW	SELECT job_title FROM jobs WHERE (max_salary + min_salary) / 2 >= ?xi?
54	OR	ROR	RORW	SELECT job_title FROM jobs WHERE (max_salary + min_salary) / 2 <= ?xi?
55	OR	ROR	RORW	SELECT job_title FROM jobs WHERE (1=1)
56	OR	ROR	RORW	SELECT job_title FROM jobs WHERE (1=0)

Query 10

SELECT postal_code from locations where location_id in(?ai?,?bi?,?ci?)

ID	Cat	Type	SubType	Mutated SQL
7	OR	ABS	ABSW	SELECT postal_code FROM locations WHERE ABS(location_id) IN (?ai? , ?bi? , ?ci?)
8	OR	ABS	ABSW	SELECT postal_code FROM locations WHERE (-ABS(location_id)) IN (?ai? , ?bi? , ?ci?)
9	OR	UOI	UOIW	SELECT postal_code FROM locations WHERE ((location_id)+1) IN (?ai? , ?bi? , ?ci?)
10	OR	UOI	UOIW	SELECT postal_code FROM locations WHERE ((location_id)-1) IN (?ai? , ?bi? , ?ci?)
11	OR	UOI	UOIW	SELECT postal_code FROM locations WHERE (-(location_id)) IN (?ai? , ?bi? , ?ci?)
15	SC	SUB	SUBQ	SELECT postal_code FROM locations WHERE location_id NOT IN (?ai? , ?bi? , ?ci?)
16	IR	IRP	IRPCW	SELECT postal_code FROM locations WHERE location_id IN (LOCATIONS.LOCATION_ID , ?bi? , ?ci?)
19	IR	IRP	IRPCW	SELECT postal_code FROM locations WHERE location_id IN (?ai? , LOCATIONS.LOCATION_ID , ?ci?)
22	IR	IRP	IRPCW	SELECT postal_code FROM locations WHERE location_id IN (?ai? , ?bi? , LOCATIONS.LOCATION_ID)

Query 11

SELECT first_name||last_name from employees where commission_pct * 100>?xi?

ID	Cat	Type	SubType	Mutated SQL
13	NL	NLI	NLIW	SELECT first_name || last_name FROM employees

				WHERE (EMPLOYEES.COMMISSION_PCT IS NULL OR commission_pct * 100 > ?xi?)
14	NL	NLO	NLIW1	SELECT first_name || last_name FROM employees WHERE (EMPLOYEES.COMMISSION_PCT IS NULL OR NOT commission_pct * 100 > ?xi?)
17	OR	ABS	ABSW	SELECT first_name || last_name FROM employees WHERE ABS(commission_pct * 100) > ?xi?
18	OR	ABS	ABSW	SELECT first_name || last_name FROM employees WHERE (- ABS(commission_pct * 100)) > ?xi?
19	OR	UOI	UOIW	SELECT first_name || last_name FROM employees WHERE ((commission_pct * 100)+1) > ?xi?
20	OR	UOI	UOIW	SELECT first_name || last_name FROM employees WHERE ((commission_pct * 100)-1) > ?xi?
21	OR	UOI	UOIW	SELECT first_name || last_name FROM employees WHERE (-(commission_pct * 100)) > ?xi?
22	OR	ABS	ABSW	SELECT first_name || last_name FROM employees WHERE ABS(commission_pct) * 100 > ?xi?
23	OR	ABS	ABSW	SELECT first_name || last_name FROM employees WHERE (-ABS(commission_pct)) * 100 > ?xi?
24	OR	UOI	UOIW	SELECT first_name || last_name FROM employees WHERE ((commission_pct)+1) * 100 > ?xi?
25	OR	UOI	UOIW	SELECT first_name || last_name FROM employees WHERE ((commission_pct)-1) * 100 > ?xi?
26	OR	UOI	UOIW	SELECT first_name || last_name FROM employees WHERE (-(commission_pct)) * 100 > ?xi?
28	IR	IRC	IRCTW	SELECT first_name || last_name FROM employees WHERE 100 * 100 > ?xi?
29	IR	IRD	IRDDW	SELECT first_name || last_name FROM employees WHERE EMPLOYEES.EMPLOYEE_ID * 100 > ?xi?
30	IR	IRD	IRDDW	SELECT first_name || last_name FROM employees WHERE EMPLOYEES.SALARY * 100 > ?xi?
31	IR	IRD	IRDDW	SELECT first_name || last_name FROM employees WHERE EMPLOYEES.MANAGER_ID * 100 > ?xi?
32	IR	IRD	IRDDW	SELECT first_name || last_name FROM employees WHERE EMPLOYEES.DEPARTMENT_ID * 100 > ?xi?
33	OR	AOR	AORW	SELECT first_name || last_name FROM employees

				WHERE commission_pct + 100 > ?xi?
34	OR	AOR	AORW	SELECT first_name || last_name FROM employees WHERE commission_pct - 100 > ?xi?
35	OR	AOR	AORW	SELECT first_name || last_name FROM employees WHERE commission_pct / 100 > ?xi?
36	OR	AOR	AORW	SELECT first_name || last_name FROM employees WHERE commission_pct > ?xi?
37	OR	AOR	AORW	SELECT first_name || last_name FROM employees WHERE 100 > ?xi?
38	IR	IRT	IRTCW	SELECT first_name || last_name FROM employees WHERE commission_pct * EMPLOYEES.COMMISSION_PCT > ?xi?
39	IR	IRT	IRTPW	SELECT first_name || last_name FROM employees WHERE commission_pct * ?xi? > ?xi?
41	OR	ROR	RORW	SELECT first_name || last_name FROM employees WHERE commission_pct * 100 <> ?xi?
42	OR	ROR	RORW	SELECT first_name || last_name FROM employees WHERE commission_pct * 100 < ?xi?
43	OR	ROR	RORW	SELECT first_name || last_name FROM employees WHERE commission_pct * 100 >= ?xi?
44	OR	ROR	RORW	SELECT first_name || last_name FROM employees WHERE commission_pct * 100 <= ?xi?
45	OR	ROR	RORW	SELECT first_name || last_name FROM employees WHERE (1=1)

Query 12

SELECT location_id from locations where state_province is not null and street_address=?xc?

ID	Cat	Type	SubType	Mutated SQL
7	NL	NLI	NLIW	SELECT location_id FROM locations WHERE (state_province IS NOT NULL) AND (LOCATIONS.STREET_ADDRESS IS NULL OR

				street_address = ?xc?)
8	NL	NLO	NLIW1	SELECT location_id FROM locations WHERE (state_province IS NOT NULL) AND (LOCATIONS.STREET_ADDRESS IS NULL OR NOT street_address = ?xc?)
11	NL	NLF	NLFW	SELECT location_id FROM locations WHERE (state_province IS NULL) AND street_address = ?xc?
12	IR	IRC	IRCCW	SELECT location_id FROM locations WHERE (LOCATIONS.STREET_ADDRESS IS NOT NULL) AND street_address = ?xc?
14	IR	IRD	IRDDW	SELECT location_id FROM locations WHERE (LOCATIONS.POSTAL_CODE IS NOT NULL) AND street_address = ?xc?
15	IR	IRD	IRDDW	SELECT location_id FROM locations WHERE (LOCATIONS.CITY IS NOT NULL) AND street_address = ?xc?
16	IR	IRD	IRDDW	SELECT location_id FROM locations WHERE (LOCATIONS.COUNTRY_ID IS NOT NULL) AND street_address = ?xc?
17	OR	LCR	LCRW	SELECT location_id FROM locations WHERE (state_province IS NOT NULL) OR street_address = ?xc?
21	OR	LCR	LCRW	SELECT location_id FROM locations WHERE street_address = ?xc?
22	IR	IRC	IRCCW	SELECT location_id FROM locations WHERE (state_province IS NOT NULL) AND LOCATIONS.STATE_PROVINCE = ?xc?
23	IR	IRD	IRDDW	SELECT location_id FROM locations WHERE (state_province IS NOT NULL) AND LOCATIONS.POSTAL_CODE = ?xc?
24	IR	IRD	IRDDW	SELECT location_id FROM locations WHERE (state_province IS NOT NULL) AND LOCATIONS.CITY = ?xc?
25	IR	IRD	IRDDW	SELECT location_id FROM locations WHERE (state_province IS NOT NULL) AND LOCATIONS.COUNTRY_ID = ?xc?
26	OR	ROR	RORW	SELECT location_id FROM locations WHERE (

				state_province IS NOT NULL) AND street_address <> ?xc?
27	OR	ROR	RORW	SELECT location_id FROM locations WHERE (state_province IS NOT NULL) AND street_address > ?xc?
28	OR	ROR	RORW	SELECT location_id FROM locations WHERE (state_province IS NOT NULL) AND street_address < ?xc?
29	OR	ROR	RORW	SELECT location_id FROM locations WHERE (state_province IS NOT NULL) AND street_address >= ?xc?
30	OR	ROR	RORW	SELECT location_id FROM locations WHERE (state_province IS NOT NULL) AND street_address <= ?xc?

Query 13

SELECT * from departments d,locations l where d.location_id=l.location_id

and l.city like ?xc?

ID	Cat	Type	SubType	Mutated SQL
1	SC	SEL	SLCT	SELECT DISTINCT * FROM departments d , locations l WHERE d.location_id = l.location_id AND l.city LIKE ?xc?
3	SC	JOI	JOIN	SELECT * FROM departments d RIGHT JOIN locations l ON d.location_id = l.location_id WHERE l.city LIKE ?xc?
4	SC	JOI	JOIN	SELECT * FROM departments d FULL OUTER JOIN locations l ON d.location_id = l.location_id WHERE l.city LIKE ?xc?
5	NL	NLI	NLIW	SELECT * FROM departments d , locations l WHERE (d.LOCATION_ID IS NULL OR d.location_id = l.location_id) AND l.city LIKE ?xc?
6	NL	NLO	NLIW1	SELECT * FROM departments d , locations l WHERE (d.LOCATION_ID IS NULL OR NOT d.location_id =

				l.location_id) AND l.city LIKE ?xc?
7	NL	NLO	NLIW2	SELECT * FROM departments d , locations l WHERE (d.LOCATION_ID IS NULL) AND l.city LIKE ?xc?
8	NL	NLO	NLIW3	SELECT * FROM departments d , locations l WHERE (d.LOCATION_ID IS NOT NULL) AND l.city LIKE ?xc?
9	OR	ABS	ABSW	SELECT * FROM departments d , locations l WHERE ABS(d.location_id) = l.location_id AND l.city LIKE ?xc?
10	OR	ABS	ABSW	SELECT * FROM departments d , locations l WHERE (-ABS(d.location_id)) = l.location_id AND l.city LIKE ?xc?
11	OR	UOI	UOIW	SELECT * FROM departments d , locations l WHERE ((d.location_id)+1) = l.location_id AND l.city LIKE ?xc?
12	OR	UOI	UOIW	SELECT * FROM departments d , locations l WHERE ((d.location_id)-1) = l.location_id AND l.city LIKE ?xc?
13	OR	UOI	UOIW	SELECT * FROM departments d , locations l WHERE (-(d.location_id)) = l.location_id AND l.city LIKE ?xc?
14	IR	IRD	IRDDW	SELECT * FROM departments d , locations l WHERE d.DEPARTMENT_ID = l.location_id AND l.city LIKE ?xc?
15	IR	IRD	IRDDW	SELECT * FROM departments d , locations l WHERE d.MANAGER_ID = l.location_id AND l.city LIKE ?xc?
16	OR	ROR	RORW	SELECT * FROM departments d , locations l WHERE d.location_id <> l.location_id AND l.city LIKE ?xc?
17	OR	ROR	RORW	SELECT * FROM departments d , locations l WHERE d.location_id > l.location_id AND l.city LIKE ?xc?
18	OR	ROR	RORW	SELECT * FROM departments d , locations l WHERE d.location_id < l.location_id AND l.city LIKE ?xc?
19	OR	ROR	RORW	SELECT * FROM departments d , locations l WHERE d.location_id >= l.location_id AND l.city LIKE ?xc?
20	OR	ROR	RORW	SELECT * FROM departments d , locations l WHERE d.location_id <= l.location_id AND l.city LIKE ?xc?

21	OR	ROR	RORW	SELECT * FROM departments d , locations l WHERE (1=1) AND l.city LIKE ?xc?
22	OR	ROR	RORW	SELECT * FROM departments d , locations l WHERE (1=0) AND l.city LIKE ?xc?
23	OR	ABS	ABSW	SELECT * FROM departments d , locations l WHERE d.location_id = ABS(l.location_id) AND l.city LIKE ?xc?
24	OR	ABS	ABSW	SELECT * FROM departments d , locations l WHERE d.location_id = (-ABS(l.location_id)) AND l.city LIKE ?xc?
25	OR	UOI	UOIW	SELECT * FROM departments d , locations l WHERE d.location_id = ((l.location_id)+1) AND l.city LIKE ?xc?
26	OR	UOI	UOIW	SELECT * FROM departments d , locations l WHERE d.location_id = ((l.location_id)-1) AND l.city LIKE ?xc?
27	OR	UOI	UOIW	SELECT * FROM departments d , locations l WHERE d.location_id = (-(l.location_id)) AND l.city LIKE ?xc?
28	OR	LCR	LCRW	SELECT * FROM departments d , locations l WHERE d.location_id = l.location_id OR l.city LIKE ?xc?
29	OR	LCR	LCRW	SELECT * FROM departments d , locations l WHERE (1=1)
30	OR	LCR	LCRW	SELECT * FROM departments d , locations l WHERE (1=0)
31	OR	LCR	LCRW	SELECT * FROM departments d , locations l WHERE d.location_id = l.location_id
32	OR	LCR	LCRW	SELECT * FROM departments d , locations l WHERE l.city LIKE ?xc?
33	IR	IRC	IRCPW	SELECT * FROM departments d , locations l WHERE d.location_id = l.location_id AND ?xc? LIKE ?xc?
34	IR	IRD	IRDDW	SELECT * FROM departments d , locations l WHERE d.location_id = l.location_id AND l.STREET_ADDRESS LIKE ?xc?
35	IR	IRD	IRDDW	SELECT * FROM departments d , locations l WHERE d.location_id = l.location_id AND l.POSTAL_CODE LIKE ?xc?

Query 14

SELECT job_title,max(salary) from employees e, jobs j where

e.job_id=j.job_id group by job_title having max(salary)>?xi?

ID	Cat	Type	SubType	Mutated SQL
2	NL	NLS	NLSS	SELECT job_title , COALESCE(MAX(salary) ,9999) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
3	SC	AGR	AGRS	SELECT job_title , COUNT(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
4	SC	AGR	AGRS	SELECT job_title , COUNT(DISTINCT salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
5	SC	AGR	AGRS	SELECT job_title , SUM(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
6	SC	AGR	AGRS	SELECT job_title , SUM(DISTINCT salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
7	SC	AGR	AGRS	SELECT job_title , MIN(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
8	SC	AGR	AGRS	SELECT job_title , AVG(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
9	SC	AGR	AGRS	SELECT job_title , AVG(DISTINCT salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
10	OR	ABS	ABSS	SELECT job_title , MAX(ABS(salary)) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
11	OR	ABS	ABSS	SELECT job_title , MAX((-ABS(salary))) FROM

				employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
12	OR	UOI	UOIS	SELECT job_title , MAX(((salary)+1)) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
13	OR	UOI	UOIS	SELECT job_title , MAX(((salary)-1)) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
14	OR	UOI	UOIS	SELECT job_title , MAX((-(salary))) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
15	IR	IRC	IRCPS	SELECT job_title , MAX(?xi?) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
16	IR	IRD	IRDDS	SELECT job_title , MAX(e.EMPLOYEE_ID) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
17	IR	IRD	IRDDS	SELECT job_title , MAX(e.COMMISSION_PCT) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
18	IR	IRD	IRDDS	SELECT job_title , MAX(e.MANAGER_ID) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
19	IR	IRD	IRDDS	SELECT job_title , MAX(e.DEPARTMENT_ID) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
20	SC	JOI	JOIN	SELECT job_title , MAX(salary) FROM employees e LEFT JOIN jobs j ON e.job_id = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
22	SC	JOI	JOIN	SELECT job_title , MAX(salary) FROM employees e FULL OUTER JOIN jobs j ON e.job_id = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
23	IR	IRC	IRCCW	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE j.JOB_TITLE = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?

24	IR	IRD	IRDDW	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.FIRST_NAME = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
25	IR	IRD	IRDDW	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.LAST_NAME = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
26	IR	IRD	IRDDW	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.EMAIL = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
27	IR	IRD	IRDDW	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.PHONE_NUMBER = j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
28	OR	ROR	RORW	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id <> j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
29	OR	ROR	RORW	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id > j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
30	OR	ROR	RORW	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id < j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
31	OR	ROR	RORW	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id >= j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
32	OR	ROR	RORW	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id <= j.job_id GROUP BY job_title HAVING MAX(salary) > ?xi?
33	OR	ROR	RORW	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE (1=1) GROUP BY job_title HAVING MAX(salary) > ?xi?
34	OR	ROR	RORW	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE (1=0) GROUP BY job_title HAVING MAX(salary) > ?xi?
35	IR	IRC	IRCCW	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.JOB_TITLE GROUP BY job_title HAVING MAX(salary) > ?xi?
36	SC	GRU	GRUP	SELECT MIN(job_title) AS job_title , MAX(salary)

				FROM employees e , jobs j WHERE e.job_id = j.job_id
37	SC	GRU	GRUP	SELECT MAX(job_title) AS job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id
38	IR	IRC	IRCCG	SELECT e.JOB_ID , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY e.JOB_ID HAVING MAX(salary) > ?xi?
39	IR	IRC	IRCCG	SELECT j.JOB_ID , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY j.JOB_ID HAVING MAX(salary) > ?xi?
40	NL	NLI	NLIH	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING (MAX(e.SALARY) IS NULL OR MAX(salary) > ?xi?)
41				SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING (MAX(e.SALARY) IS NULL OR NOT MAX(salary) > ?xi?)
42				SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING (MAX(e.SALARY) IS NULL)
43				SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING (MAX(e.SALARY) IS NOT NULL)
44	SC	AGR	AGRH	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING COUNT(salary) > ?xi?
45	SC	AGR	AGRH	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING COUNT(DISTINCT salary) > ?xi?
46	SC	AGR	AGRH	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING SUM(salary) > ?xi?
47	SC	AGR	AGRH	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY

				job_title HAVING SUM(DISTINCT salary) > ?xi?
48	SC	AGR	AGRH	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MIN(salary) > ?xi?
49	SC	AGR	AGRH	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING AVG(salary) > ?xi?
50	SC	AGR	AGRH	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING AVG(DISTINCT salary) > ?xi?
51	OR	ABS	ABSH	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(ABS(salary)) > ?xi?
52	OR	ABS	ABSH	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX((-ABS(salary))) > ?xi?
53	OR	UOI	UOIH	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(((salary)+1)) > ?xi?
54	OR	UOI	UOIH	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(((salary)-1)) > ?xi?
55	OR	UOI	UOIH	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX((-(salary))) > ?xi?
56	IR	IRC	IRCPH	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(?xi?) > ?xi?
57	IR	IRD	IRDDH	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(e.EMPLOYEE_ID) > ?xi?
58	IR	IRD	IRDDH	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(e.COMMISSION_PCT) > ?xi?
59	IR	IRD	IRDDH	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY

				job_title HAVING MAX(e.MANAGER_ID) > ?xi?
60	IR	IRD	IRDDH	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(e.DEPARTMENT_ID) > ?xi?
61	OR	ROR	RORH	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(salary) = ?xi?
62	OR	ROR	RORH	SELECT job_title , MAX(salary) FROM employees e , jobs j WHERE e.job_id = j.job_id GROUP BY job_title HAVING MAX(salary) <> ?xi?

Query 15

SELECT count(*) from employees where department_id=?xi?

ID	Cat	Type	SubType	Mutated SQL
2	NL	NLI	NLIW	SELECT COUNT(*) FROM employees WHERE (EMPLOYEES.DEPARTMENT_ID IS NULL OR department_id = ?xi?)
3	NL	NLO	NLIW1	SELECT COUNT(*) FROM employees WHERE (EMPLOYEES.DEPARTMENT_ID IS NULL OR NOT department_id = ?xi?)
6	OR	ABS	ABSW	SELECT COUNT(*) FROM employees WHERE ABS(department_id) = ?xi?
7	OR	ABS	ABSW	SELECT COUNT(*) FROM employees WHERE (-ABS(department_id)) = ?xi?
8	OR	UOI	UOIW	SELECT COUNT(*) FROM employees WHERE ((department_id)+1) = ?xi?
9	OR	UOI	UOIW	SELECT COUNT(*) FROM employees WHERE ((department_id)-1) = ?xi?
10	OR	UOI	UOIW	SELECT COUNT(*) FROM employees WHERE (-(department_id)) = ?xi?
11	IR	IRD	IRDDW	SELECT COUNT(*) FROM employees WHERE EMPLOYEES.EMPLOYEE_ID = ?xi?

12	IR	IRD	IRDDW	SELECT COUNT(*) FROM employees WHERE EMPLOYEES.SALARY = ?xi?
13	IR	IRD	IRDDW	SELECT COUNT(*) FROM employees WHERE EMPLOYEES.COMMISSION_PCT = ?xi?
14	IR	IRD	IRDDW	SELECT COUNT(*) FROM employees WHERE EMPLOYEES.MANAGER_ID = ?xi?
15	OR	ROR	RORW	SELECT COUNT(*) FROM employees WHERE department_id <> ?xi?
16	OR	ROR	RORW	SELECT COUNT(*) FROM employees WHERE department_id > ?xi?
17	OR	ROR	RORW	SELECT COUNT(*) FROM employees WHERE department_id < ?xi?
18	OR	ROR	RORW	SELECT COUNT(*) FROM employees WHERE department_id >= ?xi?
19	OR	ROR	RORW	SELECT COUNT(*) FROM employees WHERE department_id <= ?xi?

Query 16

SELECT * from departments where manager_id is null and location_id=?xc?

ID	Cat	Type	SubType	Mutated SQL
2	NL	NLI	NLIW	SELECT * FROM departments WHERE (manager_id IS NULL) AND (DEPARTMENTS.LOCATION_ID IS NULL OR location_id = ?xc?)
3	NL	NLO	NLIW1	SELECT * FROM departments WHERE (manager_id IS NULL) AND (DEPARTMENTS.LOCATION_ID IS NULL OR NOT location_id = ?xc?)
6	NL	NLF	NLFW	SELECT * FROM departments WHERE (manager_id IS NOT NULL) AND location_id = ?xc?
7	IR	IRC	IRCCW	SELECT * FROM departments WHERE (DEPARTMENTS.LOCATION_ID IS NULL) AND location_id = ?xc?
9	OR	LCR	LCRW	SELECT * FROM departments WHERE (manager_id IS NULL) OR location_id = ?xc?
10	OR	LCR	LCRW	SELECT * FROM departments WHERE (1=1)

14	OR	ABS	ABSW	SELECT * FROM departments WHERE (manager_id IS NULL) AND ABS(location_id) = ?xc?
15	OR	ABS	ABSW	SELECT * FROM departments WHERE (manager_id IS NULL) AND (-ABS(location_id)) = ?xc?
16	OR	UOI	UOIW	SELECT * FROM departments WHERE (manager_id IS NULL) AND ((location_id)+1) = ?xc?
17	OR	UOI	UOIW	SELECT * FROM departments WHERE (manager_id IS NULL) AND ((location_id)-1) = ?xc?
18	OR	UOI	UOIW	SELECT * FROM departments WHERE (manager_id IS NULL) AND (-(location_id)) = ?xc?
19	IR	IRC	IRCCW	SELECT * FROM departments WHERE (manager_id IS NULL) AND DEPARTMENTS.MANAGER_ID = ?xc?
20	IR	IRD	IRDDW	SELECT * FROM departments WHERE (manager_id IS NULL) AND DEPARTMENTS.DEPARTMENT_ID = ?xc?
21	OR	ROR	RORW	SELECT * FROM departments WHERE (manager_id IS NULL) AND location_id <> ?xc?
22	OR	ROR	RORW	SELECT * FROM departments WHERE (manager_id IS NULL) AND location_id > ?xc?
23	OR	ROR	RORW	SELECT * FROM departments WHERE (manager_id IS NULL) AND location_id < ?xc?
24	OR	ROR	RORW	SELECT * FROM departments WHERE (manager_id IS NULL) AND location_id >= ?xc?
25	OR	ROR	RORW	SELECT * FROM departments WHERE (manager_id IS NULL) AND location_id <= ?xc?
26	OR	ROR	RORW	SELECT * FROM departments WHERE (manager_id IS NULL) AND (1=1)

Query 17

SELECT region_id, count(country_id) from countries where region_id>?xi? group by region_id having count(country_id)<?yi? order by region_id desc

ID	Cat	Type	SubType	Mutated SQL
14	NL	NLI	NLIW	SELECT region_id , COUNT(country_id) FROM countries WHERE (COUNTRIES.REGION_ID IS NULL OR region_id > ?xi?) GROUP BY region_id HAVING COUNT(country_id) < ?yi? ORDER BY region_id DESC
15	NL	NLO	NLIW1	SELECT region_id , COUNT(country_id) FROM countries WHERE (COUNTRIES.REGION_ID IS NULL OR NOT region_id > ?xi?) GROUP BY region_id HAVING COUNT(country_id) < ?yi? ORDER BY region_id DESC
16	NL	NLO	NLIW2	SELECT region_id , COUNT(country_id) FROM countries WHERE (COUNTRIES.REGION_ID IS NULL) GROUP BY region_id HAVING COUNT(country_id) < ?yi? ORDER BY region_id DESC
17	NL	NLO	NLIW3	SELECT region_id , COUNT(country_id) FROM countries WHERE (COUNTRIES.REGION_ID IS NOT NULL) GROUP BY region_id HAVING COUNT(country_id) < ?yi? ORDER BY region_id DESC
18	OR	ABS	ABSW	SELECT region_id , COUNT(country_id) FROM countries WHERE ABS(region_id) > ?xi? GROUP BY region_id HAVING COUNT(country_id) < ?yi? ORDER BY region_id DESC
19	OR	ABS	ABSW	SELECT region_id , COUNT(country_id) FROM countries WHERE (-ABS(region_id)) > ?xi? GROUP BY region_id HAVING COUNT(country_id) < ?yi? ORDER BY region_id DESC
20	OR	UOI	UOIW	SELECT region_id , COUNT(country_id) FROM countries WHERE ((region_id)+1) > ?xi? GROUP BY region_id HAVING COUNT(country_id) < ?yi? ORDER BY region_id DESC
21	OR	UOI	UOIW	SELECT region_id , COUNT(country_id) FROM countries WHERE ((region_id)-1) > ?xi? GROUP BY region_id HAVING COUNT(country_id) < ?yi? ORDER BY region_id DESC
22	OR	UOI	UOIW	SELECT region_id , COUNT(country_id) FROM countries WHERE (-(region_id)) > ?xi? GROUP BY

				region_id HAVING COUNT(country_id) < ?yi? ORDER BY region_id DESC
23	OR	ROR	RORW	SELECT region_id , COUNT(country_id) FROM countries WHERE region_id = ?xi? GROUP BY region_id HAVING COUNT(country_id) < ?yi? ORDER BY region_id DESC
24	OR	ROR	RORW	SELECT region_id , COUNT(country_id) FROM countries WHERE region_id <> ?xi? GROUP BY region_id HAVING COUNT(country_id) < ?yi? ORDER BY region_id DESC
25	OR	ROR	RORW	SELECT region_id , COUNT(country_id) FROM countries WHERE region_id < ?xi? GROUP BY region_id HAVING COUNT(country_id) < ?yi? ORDER BY region_id DESC
26	OR	ROR	RORW	SELECT region_id , COUNT(country_id) FROM countries WHERE region_id >= ?xi? GROUP BY region_id HAVING COUNT(country_id) < ?yi? ORDER BY region_id DESC
27	OR	ROR	RORW	SELECT region_id , COUNT(country_id) FROM countries WHERE region_id <= ?xi? GROUP BY region_id HAVING COUNT(country_id) < ?yi? ORDER BY region_id DESC
28	OR	ROR	RORW	SELECT region_id , COUNT(country_id) FROM countries WHERE (1=1) GROUP BY region_id HAVING COUNT(country_id) < ?yi? ORDER BY region_id DESC
29	OR	ROR	RORW	SELECT region_id , COUNT(country_id) FROM countries WHERE (1=0) GROUP BY region_id HAVING COUNT(country_id) < ?yi? ORDER BY region_id DESC
33	SC	AGR	AGRH	SELECT region_id , COUNT(country_id) FROM countries WHERE region_id > ?xi? GROUP BY region_id HAVING MAX(country_id) < ?yi? ORDER BY region_id DESC
34	SC	AGR	AGRH	SELECT region_id , COUNT(country_id) FROM countries WHERE region_id > ?xi? GROUP BY region_id HAVING MIN(country_id) < ?yi? ORDER BY region_id DESC

35	IR	IRD	IRDDH	SELECT region_id , COUNT(country_id) FROM countries WHERE region_id > ?xi? GROUP BY region_id HAVING COUNT(COUNTRIES.COUNTRY_NAME) < ?yi? ORDER BY region_id DESC
36	OR	ROR	RORH	SELECT region_id , COUNT(country_id) FROM countries WHERE region_id > ?xi? GROUP BY region_id HAVING COUNT(country_id) = ?yi? ORDER BY region_id DESC
37	OR	ROR	RORH	SELECT region_id , COUNT(country_id) FROM countries WHERE region_id > ?xi? GROUP BY region_id HAVING COUNT(country_id) <> ?yi? ORDER BY region_id DESC
38	OR	ROR	RORH	SELECT region_id , COUNT(country_id) FROM countries WHERE region_id > ?xi? GROUP BY region_id HAVING COUNT(country_id) > ?yi? ORDER BY region_id DESC
39	OR	ROR	RORH	SELECT region_id , COUNT(country_id) FROM countries WHERE region_id > ?xi? GROUP BY region_id HAVING COUNT(country_id) >= ?yi? ORDER BY region_id DESC
40	OR	ROR	RORH	SELECT region_id , COUNT(country_id) FROM countries WHERE region_id > ?xi? GROUP BY region_id HAVING COUNT(country_id) <= ?yi? ORDER BY region_id DESC
41	OR	ROR	RORH	SELECT region_id , COUNT(country_id) FROM countries WHERE region_id > ?xi? GROUP BY region_id HAVING (1=1) ORDER BY region_id DESC
42	OR	ROR	RORH	SELECT region_id , COUNT(country_id) FROM countries WHERE region_id > ?xi? GROUP BY region_id HAVING (1=0) ORDER BY region_id DESC

Appendix B

HR Tables Descriptions

Tables name:

Table COUNTRIES

Name	Null?	Type
COUNTRY_ID	NOT NULL	CHAR(2)
COUNTRY_NAME		VARCHAR2(40)
REGION_ID		NUMBER

Table DEPARTMENTS

Name	Null?	Type
DEPARTMENT_ID	NOT NULL	NUMBER(4)
DEPARTMENT_NAME	NOT NULL	VARCHAR2(30)
MANAGER_ID		NUMBER(6)
LOCATION_ID		NUMBER(4)

Table EMPLOYEES

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
FIRST_NAME		VARCHAR2(20)
LAST_NAME	NOT NULL	VARCHAR2(25)
EMAIL	NOT NULL	VARCHAR2(25)
PHONE_NUMBER		VARCHAR2(20)
HIRE_DATE	NOT NULL	DATE

JOB_ID	NOT NULL	VARCHAR2(10)
SALARY		NUMBER(8,2)
COMMISSION_PCT		NUMBER(2,2)
MANAGER_ID		NUMBER(6)
DEPARTMENT_ID		NUMBER(4)

Table JOBS

Name	Null?	Type
JOB_ID	NOT NULL	VARCHAR2(10)
JOB_TITLE	NOT NULL	VARCHAR2(35)
MIN_SALARY		NUMBER(6)
MAX_SALARY		NUMBER(6)

Table JOB_HISTORY

Name	Null?	Type
EMPLOYEE_ID	NOT NULL	NUMBER(6)
START_DATE	NOT NULL	DATE
END_DATE	NOT NULL	DATE
JOB_ID	NOT NULL	VARCHAR2(10)
DEPARTMENT_ID		NUMBER(4)

Table LOCATIONS

Name	Null?	Type
LOCATION_ID	NOT NULL	NUMBER(4)
STREET_ADDRESS		VARCHAR2(40)
POSTAL_CODE		VARCHAR2(12)
CITY	NOT NULL	VARCHAR2(30)

STATE_PROVINCE		VARCHAR2(25)
COUNTRY_ID		CHAR(2)

Table REGIONS

Name	Null?	Type
-------------------------------	------------------------------------	------------------------------------
REGION_ID	NOT NULL	NUMBER
REGION_NAME		VARCHAR2(25)

Appendix C

GA with Fitness Table result

Query 2

SELECT employee_id from employees where hire_date>?xu? order by department_id

Table 33 Query 2 Mutant's Exceptions Coverage

Mutant ID	Mutant Subtype	Exceptions						GPN
		NDF		TMR		Others		
		GN	Coverage	GN	Coverage	GN	Coverage	
11	RORW	1	y	50	n	1	y	3
12	RORW	50	n	1	y	1	y	2
13	RORW	1	y	1	y	1	y	1
14	RORW	1	y	1	y	1	y	1
15	RORW	1	y	1	y	1	y	1

Query 3

SELECT manager_id from departments where location_id=?xi? or department_id=?xi?

Table 34 Query 3 Mutant's Exceptions Coverage

Mutant ID	Mutant Subtype	Exceptions						GPN
		NDF		TMR		Others		
		GN	Coverage	GN	Coverage	GN	Coverage	
12	NLIW	1	y	50	n	1	y	3
13	NLIW1	50	n	1	y	1	y	2
14	NLIW2	1	y	50	n	1	y	2
15	NLIW3	50	n	1	y	1	y	2

16	ABSW	1	y	48	y	1	y	1
17	ABSW	1	y	50	n	1	y	2
18	UOI	1	y	50	n	1	y	3
19	UOI	1	y	40	n	1	y	3
20	UOI	1	y	50	n	1	y	2
21	IRCCW	1	y	50	n	1	y	2
22	IRCCW	1	y	50	n	1	y	2
23	RORW	50	n	1	y	1	y	2
24	RORW	1	y	1	y	1	y	1
25	RORW	5	y	1	y	1	y	1
26	RORW	1	y	2	y	1	y	1
27	RORW	9	y	1	y	1	y	1
28	RORW	50	n	1	y	1	y	2
29	RORW	1	y	50	n	1	y	2
30	IRPCW	1	y	50	n	1	y	2
31	IRPCW	1	y	50	n	1	y	2
33	LCRW	1	y	50	n	1	y	2
36	LCRW	1	y	41	n	1	y	3
37	LCRW	1	y	50	n	1	y	2

Query 4

SELECT e.employee_id,j.max_salary from employees e, jobs j where e.job_id=j.job_id and e.commission_pct<=?xd?

Table 35 Query 4 Mutant's Exceptions Coverage

Mutant ID	Mutant Subtype	Exceptions						GPN
		NDF		TMR		Others		
		GN	Coverage	GN	Coverage	GN	Coverage	
23	NLIW	50	n	1	y	1	y	2
24	NLIW1	50	n	1	y	1	y	2
27	IRDDW	1	y	50	n	1	y	2
28	IRDDW	1	y	50	n	1	y	2
29	IRDDW	1	y	50	n	1	y	2
30	IRDDW	1	y	50	n	1	y	2

31	RORW	1	y	50	n	1	y	3
32	RORW	1	y	1	y	1	y	1
33	RORW	1	y	1	y	1	y	1
34	RORW	1	y	1	y	1	y	1
35	RORW	1	y	1	y	1	y	1
36	RORW	1	y	1	y	1	y	1
37	RORW	1	y	50	n	1	y	2
38	IRDDW	1	y	50	n	1	y	2
39	LCRW	50	n	1	y	1	y	2
43	LCRW	1	y	1	y	1	y	1
44	ABSW	1	y	1	y	1	y	1
45	ABSW	50	n	1	y	1	y	2
46	UOIW	1	y	50	n	1	y	2
47	UOIW	50	n	1	y	1	y	2
48	UOIW	50	n	1	y	1	y	2
49	IRCCW	1	y	50	n	1	y	2
50	IRCCW	1	y	50	n	1	y	2
51	IRCCW	1	y	50	n	1	y	2
52	IRCCW	1	y	50	n	1	y	2
53	IRCCW	1	y	50	n	1	y	3
54	RORW	1	y	1	y	1	y	1
55	RORW	2	y	1	y	1	y	1
56	RORW	1	y	1	y	1	y	1
57	RORW	1	y	1	y	1	y	1
58	RORW	1	y	1	y	1	y	1

Query 5

SELECT e.first_name from employees e left join job_history h on e.employee_id=h.employee_id and h.start_date=?xu?

Table 36 Query 5 Mutant's Exceptions Coverage

Mutant ID	Mutant Subtype	Exceptions						GPN
		NDF		TMR		Others		
		G	Coverag	G	Coverag	G	Coverag	

		N	e	N	e	N	e	
7	JOIN	50	n	1	y	1	y	2
8	JOIN	1	y	50	n	1	y	3
9	JOIN	50	n	1	y	1	y	2
11	ABSJ	50	n	1	y	1	y	2
12	ABSJ	50	n	1	y	1	y	2
13	UOIJ	50	n	1	y	1	y	2
14	UOIJ	50	n	1	y	1	y	2
15	UOIJ	50	n	1	y	1	y	2
16	IRDDJ	50	n	1	y	1	y	2
17	IRDDJ	50	n	1	y	1	y	2
18	IRDDJ	50	n	1	y	1	y	2
19	IRDDJ	50	n	1	y	1	y	2
20	RORJ	50	n	1	y	1	y	2
21	RORJ	50	n	1	y	1	y	2
22	RORJ	50	n	1	y	1	y	2
23	RORJ	50	n	1	y	1	y	2
24	RORJ	50	n	1	y	1	y	2
25	RORJ	50	n	1	y	1	y	2
26	RORJ	50	n	1	y	1	y	2
27	ABSJ	50	n	1	y	1	y	2
28	ABSJ	50	n	1	y	1	y	2
29	UOIJ	50	n	1	y	1	y	2
30	UOIJ	50	n	1	y	1	y	2
31	UOIJ	50	n	1	y	1	y	2
32	IRDDJ	50	n	1	y	1	y	2
33	LCRJ	50	n	1	y	1	y	2
37	LCRJ	50	n	1	y	1	y	2
38	IRDDJ	50	n	1	y	1	y	2
39	RORJ	50	n	1	y	1	y	2
40	RORJ	50	n	1	y	1	y	2
41	RORJ	50	n	1	y	1	y	2
42	RORJ	50	n	1	y	1	y	2
43	RORJ	50	n	1	y	1	y	2

Query 6

SELECT job_id,avg(salary) from employees group by job_id having sum(salary)<?xi?

Table 37 Query 6 Mutant's Exceptions Coverage

Mutant ID	Mutant Subtype	Exceptions						GPN
		NDF		TMR		Others		
		GN	Coverage	GN	Coverage	GN	Coverage	
26	NLIH	2	y	1	y	1	y	1
27	NLIH	50	n	1	y	1	y	2
30	AGRH	43	n	1	y	1	y	3
31	AGRH	4	y	10	y	1	y	1
32	AGRH	1	y	1	y	1	y	1
33	AGRH	1	y	1	y	1	y	1
34	AGRH	3	y	1	y	1	y	1
35	AGRH	1	y	1	y	1	y	1
36	AGRH	2	y	1	y	1	y	1
37	ABSH	1	y	1	y	1	y	1
38	ABSH	50	n	1	y	1	y	2
39	UOIH	1	y	1	y	1	y	1
40	UOIH	2	y	1	y	1	y	1
41	UOIH	50	n	1	y	1	y	2
43	IRCPH	50	n	1	y	1	y	3
44	IRCPH	50	n	1	y	1	y	3
45	IRCPH	26	y	1	y	1	y	1
46	IRCPH	50	n	1	y	1	y	3
47	RORH	1	y	16	y	1	y	1
48	RORH	50	n	1	y	1	y	2
49	RORH	50	n	1	y	1	y	2
50	RORH	50	n	1	y	1	y	2
51	RORH	1	y	1	y	1	y	1

Query 7

SELECT job_id from job_history where end_date between ?xu? and ?yu?

Table 38 Query 7 Mutant's Exceptions Coverage

Mutant ID	Mutant Subtype	Exceptions						GPN
		NDF		TMR		Others		
		GN	Coverage	GN	Coverage	GN	Coverage	
4	IRDDW	1	y	1	y	1	y	1
5	BTWW	5	y	1	y	1	y	1
7	BTWW	1	y	2	y	1	y	1
8	BTWW	1	y	1	y	1	y	1
9	IRPCW	1	y	1	y	1	y	1
11	IRPCW	1	y	1	y	1	y	1

Query 8

SELECT region_name from countries c , regions r where
r.region_id=c.region_id and
c.country_id=?xc?

Table 39 Query 8 Mutant's Exceptions Coverage

Mutant ID	Mutant Subtype	Exceptions						GPN
		NDF		TMR		Others		
		GN	Coverage	GN	Coverage	GN	Coverage	
5	NLIW	1	y	50	n	1	y	2
6	NLIW	1	y	20	y	1	y	1
7	NLIW	1	y	50	n	1	y	2
8	NLIW	1	y	44	y	1	y	1
9	ABSW	1	y	50	n	1	y	2
10	ABSW	1	y	50	n	1	y	2
11	UOIW	1	y	50	n	1	y	2

12	UOIW	1	y	50	n	1	y	2
13	UOIW	1	y	50	n	1	y	2
14	RORW	1	y	48	y	1	y	1
15	RORW	1	y	50	n	1	y	3
16	RORW	1	y	50	n	1	y	3
17	RORW	1	y	45	y	1	y	1
18	RORW	1	y	50	n	1	y	3
19	RORW	1	y	41	y	1	y	1
20	RORW	1	y	50	n	1	y	2
21	ABSW	1	y	50	n	1	y	2
22	ABSW	1	y	50	n	1	y	2
23	UOIW	1	y	50	n	1	y	2
24	UOIW	1	y	50	n	1	y	2
25	UOIW	1	y	50	n	1	y	2
26	LCRW	50	n	1	y	1	y	2
30	LCRW	1	y	20	y	1	y	1
31	IRCCW	1	y	50	n	1	y	3
32	IRCCW	1	y	50	n	1	y	2
33	RORW	9	y	1	y	1	y	1
34	RORW	1	y	1	y	1	y	1
35	RORW	1	y	1	y	1	y	1
36	RORW	1	y	1	y	1	y	1
37	RORW	1	y	1	y	1	y	1

Query 9

SELECT job_title from jobs where (max_salary + min_salary)/2 <?xi?

Table 40 Query 9 Mutant's Exceptions Coverage

Mutant ID	Mutant Subtype	Exceptions						GPN
		NDF		TMR		Others		
		GN	Coverage	GN	Coverage	GN	Coverage	
3	NLIW	1	y	1	y	1	y	1
4	NLIW1	1	y	1	y	1	y	1

7	NLIW	2	y	1	y	1	y	1
8	NLIW1	1	y	1	y	1	y	1
11	ABSW	2	y	1	y	1	y	1
12	ABSW	50	n	1	y	1	y	2
13	UOIW	1	y	1	y	1	y	1
14	UOIW	1	y	1	y	1	y	1
15	UOIW	50	n	1	y	1	y	2
16	ABSW	3	y	1	y	1	y	1
17	ABSW	50	n	1	y	1	y	2
18	UOIW	5	y	1	y	1	y	1
19	UOIW	1	y	1	y	1	y	1
20	UOIW	50	n	1	y	1	y	2
21	ABSW	1	y	1	y	1	y	1
22	ABSW	50	n	1	y	1	y	2
23	UOIW	1	y	1	y	1	y	1
24	UOIW	2	y	1	y	1	y	1
25	UOIW	50	n	1	y	1	y	2
26	IRCCW	1	y	1	y	1	y	1
28	IRCTW	1	y	1	y	1	y	1
29	AORW	2	y	1	y	1	y	1
30	AORW	1	y	50	n	1	y	2
31	AORW	50	n	1	y	1	y	2
32	AORW	50	n	1	y	1	y	2
33	AORW	3	y	1	y	1	y	1
34	AORW	2	y	1	y	1	y	1
35	ABSW	2	y	1	y	1	y	1
36	ABSW	2	y	1	y	1	y	1
37	UOIW	1	y	1	y	1	y	1
38	UOIW	1	y	1	y	1	y	1
39	UOIW	8	y	1	y	1	y	1
40	IRCCW	2	y	1	y	1	y	1
42	IRCTW	2	y	1	y	1	y	1
43	AORW	2	y	1	y	1	y	1
44	AORW	1	y	1	y	1	y	1
45	AORW	1	y	1	y	1	y	1
46	AORW	50	n	1	y	1	y	1
47	AORW	1	y	1	y	1	y	1
48	AORW	50	n	1	y	1	y	3

49	IRTCW	47	y	1	y	1	y	1
50	IRTCW	50	n	1	y	1	y	3
52	RORW	1	y	50	n	1	y	3
53	RORW	50	n	1	y	1	y	2
54	RORW	1	y	1	y	1	y	1
55	RORW	1	y	1	y	1	y	1
56	RORW	2	y	1	y	1	y	1

Query 10

SELECT postal_code from locations where location_id in(?ai?,?bi?,?ci?)

Table 41 Query 10 Mutant's Exceptions Coverage

Mutant ID	Mutant Subtype	Exceptions						GPN
		NDF		TMR		Others		
		GN	Coverage	GN	Coverage	GN	Coverage	
7	ABSW	1	y	50	n	1	y	3
8	ABSW	1	y	50	n	1	y	2
9	UOIW	1	y	50	n	1	y	3
10	UOIW	1	y	50	n	1	y	3
11	UOIW	1	y	50	n	1	y	2
15	SUBQ	50	n	1	y	1	y	2
16	IRPCW	50	n	1	y	1	y	2
19	IRPCW	50	n	1	y	1	y	2
22	IRPCW	50	n	1	y	1	y	2

Query 11

SELECT first_name||last_name from employees where commission_pct * 100>?xi?

Table 42 Query 11 Mutant's Exceptions Coverage

Mutant ID	Mutant Subtype	Exceptions						GPN
		NDF		TMR		Others		
		GN	Coverage	GN	Coverage	GN	Coverage	
13	NLIW	50	n	1	y	1	y	2
14	NLIW1	50	n	1	y	1	y	2
17	ABSW	1	y	50	n	1	y	3
18	ABSW	1	y	50	n	1	y	2
19	UOIW	1	y	40	y	1	y	1
20	UOIW	1	y	29	y	1	y	1
21	UOIW	1	y	50	n	1	y	2
22	ABSW	1	y	18	y	1	y	1
23	ABSW	1	y	50	n	1	y	2
24	UOIW	1	y	26	y	1	y	1
25	UOIW	1	y	50	n	1	y	2
26	UOIW	1	y	50	n	1	y	2
28	IRCTW	1	y	1	y	1	y	1
29	IRDDW	1	y	1	y	1	y	1
30	IRDDW	50	n	1	y	1	y	2
31	IRDDW	1	y	1	y	1	y	1
32	IRDDW	1	y	1	y	1	y	1
33	AORW	1	y	2	y	1	y	1
34	AORW	1	y	50	n	1	y	2
35	AORW	1	y	50	n	1	y	2
36	AORW	1	y	50	n	1	y	2
37	AORW	1	y	50	n	1	y	2
38	AORW	1	y	43	y	1	y	1
39	AORW	1	y	50	n	1	y	2
41	RORW	1	y	47	y	1	y	1
42	RORW	50	n	1	y	1	y	2
43	RORW	50	n	1	y	1	y	3
44	RORW	1	y	50	n	1	y	3
45	RORW	50	n	1	y	1	y	3

Query 12

SELECT location_id from locations where state_province is not null and street_address=?xc?

Table 43 Query 12 Mutant's Exceptions Coverage

Mutant ID	Mutant Subtype	Exceptions						GPN
		NDF		TMR		Others		
		GN	Coverage	GN	Coverage	GN	Coverage	
7	NLIW	1	y	50	n	1	y	2
8	NLIW1	50	n	1	y	1	y	2
11	NLFW	1	y	50	n	1	y	2
12	IRCCW	1	y	50	n	1	y	2
14	IRDDW	1	y	50	n	1	y	2
15	IRDDW	1	y	50	n	1	y	2
16	IRDDW	1	y	50	n	1	y	2
17	LCRW	50	n	1	y	1	y	2
21	LCRW	1	y	50	n	1	y	2
22	IRCCW	1	y	50	n	1	y	2
23	IRDDW	1	y	50	n	1	y	2
24	IRDDW	1	y	50	n	1	y	2
25	IRDDW	1	y	50	n	1	y	3
26	RORW	50	n	1	y	1	y	2
27	RORW	1	y	1	y	1	y	1
28	RORW	1	y	1	y	1	y	1
29	RORW	1	y	1	y	1	y	1
30	RORW	1	y	1	y	1	y	1

Query 13

SELECT * from departments d,locations l where d.location_id=l.location_id and l.city like
?xc?

Table 44 Query 13 Mutant's Exceptions Coverage

Mutant ID	Mutant Subtype	Exceptions						GPN
		NDF		TMR		Others		
		GN	Coverage	GN	Coverage	GN	Coverage	
2	NLIW	1	y	50	n	1	y	3
3	NLIW1	1	y	50	n	1	y	3
4	NLIW2	1	y	50	n	1	y	2
5	NLIW3	1	y	50	n	1	y	3
6	ABSW	1	y	50	n	1	y	3
7	ABSW	1	y	50	n	1	y	2
8	UOIW	1	y	50	n	1	y	2
9	UOIW	1	y	50	n	1	y	2
10	UOIW	1	y	50	n	1	y	2
11	IRDDW	1	y	50	n	1	y	2
12	IRDDW	1	y	50	n	1	y	2
13	RORW	1	y	37	y	1	y	1
14	RORW	1	y	34	y	1	y	1
15	RORW	1	y	50	n	1	y	2
16	RORW	1	y	50	n	1	y	3
17	RORW	1	y	6	y	1	y	1
18	RORW	1	y	50	n	1	y	3
19	RORW	1	y	50	n	1	y	2
20	ABSW	1	y	50	n	1	y	3
21	ABSW	1	y	50	n	1	y	2
22	UOIW	1	y	50	n	1	y	2
23	UOIW	1	y	50	n	1	y	2
24	UOIW	1	y	50	n	1	y	2
25	LCRW	50	n	1	y	1	y	2
29	LCRW	1	y	50	n	1	y	3
31	IRDDW	50	n	1	y	1	y	2
32	IRDDW	1	y	50	n	1	y	3
33	IRDDW	1	y	35	y	1	y	1
34	IRDDW	1	y	24	y	1	y	1
35	LKW	1	y	50	n	1	y	3

Query 14

SELECT job_title,max(salary) from employees e, jobs j where e.job_id=j.job_id group by
job_title having max(salary)>?xi?

Table 45 Query 14 Mutant's Exceptions Coverage

Mutant ID	Mutant Subtype	Exceptions						GPN
		NDF		TMR		Others		
		GN	Coverage	GN	Coverage	GN	Coverage	
20	IRCCW	1	y	50	n	1	y	2
21	IRDDW	1	y	50	n	1	y	2
22	IRDDW	1	y	50	n	1	y	2
23	IRDDW	1	y	50	n	1	y	2
24	IRDDW	1	y	50	n	1	y	2
25	RORW	1	y	1	y	1	y	1
26	RORW	1	y	1	y	1	y	1
27	RORW	1	y	1	y	1	y	1
28	RORW	1	y	1	y	1	y	1
29	RORW	1	y	1	y	1	y	1
30	RORW	1	y	1	y	1	y	1
31	RORW	1	y	50	n	1	y	2
32	IRCCW	1	y	50	n	1	y	2
37	NLH	1	y	1	y	1	y	1
38	NLH1	2	y	1	y	1	y	1
41	AGRH	1	y	47	y	1	y	1
42	AGRH	1	y	50	y	1	y	3
43	AGRH	50	n	1	y	1	y	2
44	AGRH	50	n	1	y	1	y	2
45	AGRH	1	y	1	y	1	y	1
46	AGRH	1	y	1	y	1	y	1
47	AGRH	1	y	1	y	1	y	1
48	ABSH	1	y	1	y	1	y	1
49	ABSH	1	y	50	n	1	y	2
50	UOIH	1	y	1	y	1	y	1
51	UOIH	1	y	1	y	1	y	1

52	UOIH	1	y	50	n	1	y	2
54	IRDDH	1	y	10	y	1	y	1
55	IRDDH	1	y	50	n	1	y	2
56	IRDDH	1	y	40	y	1	y	1
57	IRDDH	1	y	1	y	1	y	1
58	RORH	1	y	50	y	1	y	1
59	RORH	50	n	1	y	1	y	2
60	RORH	1	y	1	y	1	y	1
61	RORH	1	y	1	y	1	y	1
62	RORH	1	y	1	y	1	y	1

Query 15

SELECT count(*) from employees where employee_id=?xi?

Table 46 Query 15 Mutant's Exceptions Coverage

Mutant ID	Mutant Subtype	Exceptions						GPN
		NDF		TMR		Others		
		GN	Coverage	GN	Coverage	GN	Coverage	
2	NLIW	50	n	50	n	1	y	2
3	NLIW1	50	n	50	n	1	y	2
6	ABSW	50	n	50	n	1	y	2
7	ABSW	50	n	50	n	1	y	2
8	UOIW	50	n	50	n	1	y	2
9	UOIW	50	n	50	n	1	y	2
10	UOIW	50	n	50	n	1	y	2
11	IRDDW	50	n	50	n	1	y	2
12	IRDDW	50	n	50	n	1	y	2
13	IRDDW	50	n	50	n	1	y	2
14	IRDDW	50	n	50	n	1	y	2
15	RORW	50	n	50	n	1	y	2
16	RORW	50	n	50	n	1	y	2
17	RORW	50	n	50	n	1	y	2
18	RORW	50	n	50	n	1	y	2
19	RORW	50	n	50	n	1	y	2

Query 16

SELECT * from departments where manager_id is null and location_id=?xi?

Table 47 Query 16 Mutant's Exceptions Coverage

Mutant ID	Mutant Subtype	Exceptions						GPN
		NDF		TMR		Others		
		GN	Coverage	GN	Coverage	GN	Coverage	
2	NLIW	1	y	50	n	1	y	3
3	NLIW1	50	n	1	y	1	y	2
6	NLFW	1	y	48	y	1	y	1
7	IRCCW	1	y	50	n	1	y	2
9	IRDDW	1	y	50	n	1	y	2
10	LCRW	1	y	10	y	1	y	1
14	LCRW	50	n	1	y	1	y	2
15	ABSW	1	y	41	y	1	y	1
16	ABSW	1	y	50	n	1	y	2
17	UOIW	1	y	50	n	1	y	3
18	UOIW	1	y	50	n	1	y	3
19	UOIW	1	y	50	n	1	y	2
20	IRCCW	1	y	50	n	1	y	2
21	IRDDW	1	y	50	n	1	y	2
22	RORW	49	y	1	y	1	y	1
23	RORW	1	y	2	y	1	y	1
24	RORW	1	y	1	y	1	y	1
25	RORW	1	y	1	y	1	y	1
26	RORW	1	y	1	y	1	y	1

Query 17

SELECT region_id, count(country_id) from countries where region_id>?xi? group by region_id having count(country_id)<?yi? order by region_id desc

Table 48 Query 17 Mutant's Exceptions Coverage

Mutant ID	Mutant Subtype	Exceptions						GPN
		NDF		TMR		Others		
		GN	Coverage	GN	Coverage	GN	Coverage	
14	NLIW	1	y	50	n	1	y	3
15	NLIW1	50	n	1	y	1	y	3
16	NLLW2	1	y	50	n	1	y	2
17	NLIW3	30	y	1	y	1	y	1
18	ABSW	1	y	33	y	1	y	1
19	ABSW	1	y	50	n	1	y	1
20	UOIW	1	y	29	y	1	y	1
21	UOIW	1	y	50	n	1	y	3
22	UOIW	1	y	50	n	1	y	2
23	RORW	1	y	50	n	1	y	2
24	RORW	50	n	1	y	1	y	2
25	RORW	42	y	1	y	1	y	1
26	RORW	1	y	46	y	1	y	1
27	RORW	39	y	1	y	1	y	1
28	RORW	44	y	1	y	1	y	1
29	RORW	1	y	50	n	1	y	2
35	IRDDH	1	y	40	y	1	y	1
36	RORH	1	y	35	y	1	y	1
37	RORH	1	y	50	n	1	Y	3
38	RORH	1	y	50	n	1	Y	3
39	RORH	1	y	49	y	1	Y	1
40	RORH	1	y	43	y	1	Y	1
41	RORH	1	y	50	y	1	Y	3
42	RORH	1	y	50	n	1	Y	2

تغطية الاستثناءات في قاعدة البيانات المنظمة باستخدام الخوارزمية الجينية

اعداد
هبه محمد الحراحشة

المشرف
الدكتور محمد الشريدة

ملخص

تستخدم برمجيات الحاسوب في هذه الأيام في جميع المجالات. تختلف البرمجيات من حيث التعقيد والحجم. لذلك أي تطبيق بحاجة إلى التأكد من عمل وظيفته. واختبار البرمجيات يستخدم للتحقق من عمل التطبيقات بالشكل الصحيح والكشف عن وجود الأخطاء.

في هذه الرسالة أنشأنا تقنية جديدة لاختبار البرمجيات، التي تجعل حالات الاختبار آلية لتغطية ثلاثة أنواع من الاستثناءات الواردة في قاعدة بيانات أوراكل : لا يوجد بيانات، والعديد من الاستثناءات، واستثناءات أخرى. واعتمدنا في فكرتنا المقترحة الخوارزمية الجينية واختبارات الطفرات. واقترحنا استخدام دالة تحقيق الامثلية Fitness Function في الخوارزمية الجينية لعمل التجربة لتغطية ثلاثة أنواع من الاستثناءات الناتجة عن جملة SELECT وهذه الدالة تعتمد على مدى البيانات التابعة للعامود في جملة الاستعلام المستخدمة. قيم الحد الأعلى والحد الأدنى للأعمدة الموجودة في جملة الاستعلام تم اعتمادها كدالة تحقيق الأمثلية في حالة استثناء لا يوجد بيانات، بالإضافة إلى استخدام التجميع للأعمدة في جملة الاستعلام كدالة تحقيق الامثلية في استثناء العديد من الاستثناءات. تجاربنا نفذت على قاعدة بيانات مسماه بالموارد البشرية HR على قاعدة البيانات أوراكل و التي تكون موجودة على قاعدة البيانات أوراكل عند تنزيلها.

ناقشنا أيضا نتيجة كل الاستعلام وأي نوع من الاستثناءات التي تغطيها في كل جملة استعلام. وتم شرح نتيجة لكل طفرة تابعة لجملة الاستعلام فيما إذا كانت تغطي الاستثناءات أو يوجد مسارات غير مرئية. اعلي نسبة تغطية في الاستثناءات بعد تنفيذ فكرتنا (التي تعتمد على دالة تحقيق الامثلية في جمل الاستعلام) هي للاستثناءات أخرى ثم استثناء لا يوجد بيانات واقل نسبة كانت للعديد من الاستثناءات.

وأخيرا في هذه الأطروحة قارنا النتيجة مع بحث أخر قام بفحص نفس الاستثناءات بدون استخدام دالة تحقيق الامثلية. كانت النتائج مشجعة وتعطي تغطية كاملة للاستثناءات الثلاثة وبنتائج أعلى من نتائج البحث السابق.